MACHINE LEARNING APPLICATIONS FOR THE TOPOLOGY PREDICTION OF TRANSMEMBRANE BETA-BARREL PROTEINS

A dissertation submitted for the degree of
Doctor of Philosophy

School of Computing and Digital Media
London Metropolitan University

Cédric Maxime Grimaldi

January 2022

Acknowledgement

I want to acknowledge all those who had confidence in me and allowed me to accomplish this PhD thesis.

First, my thanks go to Professor Hassan Kazemian, my lead supervisor of studies. Without his valuable support, input, encouragement, and confidence in me, this work would not have been achievable. I also would like to thank my second supervisor, Dr Kenneth White, for his patience, guidance, and support. I want to thank all the School of Computing and Digital Media research colleagues at London Metropolitan University for their valuable comments and suggestions throughout this work.

Finally, I would like to thank my family for their support and patience.

Abstract

The research topic for this PhD thesis focuses on the topology prediction of beta-barrel transmembrane proteins. Transmembrane proteins adopt various conformations that are about the functions that they provide. The two most predominant classes are alpha-helix bundles and beta-barrel transmembrane proteins. Alpha-helix proteins are present in larger numbers than beta-barrel transmembrane proteins in structure databases. Therefore, there is a need to find computational tools that can predict and detect the structure of beta-barrel transmembrane proteins. Transmembrane proteins are used for active transport across the membrane or signal transduction. Knowing the importance of their roles, it becomes essential to understand the structures of the proteins. Transmembrane proteins are also a significant focus for new drug discovery. Transmembrane beta-barrel proteins play critical roles in the translocation machinery, pore formation, membrane anchoring, and ion exchange. In bioinformatics, many years of research have been spent on the topology prediction of transmembrane alpha-helices. The efforts to TMB (transmembrane beta-barrel) proteins topology prediction have been overshadowed, and the prediction accuracy could be improved with further research. Various methodologies have been developed in the past to predict TMB proteins topology. Methods developed in the literature that are available include turn identification, hydrophobicity profiles, rule-based prediction, HMM (Hidden Markov model), ANN (Artificial Neural Networks), radial basis function networks, or combinations of methods. The use of cascading classifier has never been fully explored. This research presents and evaluates approaches such as ANN (Artificial Neural Networks), KNN (K-Nearest Neighbors, SVM (Support Vector Machines), and a novel approach to TMB topology prediction with the use of a cascading classifier. Computer simulations have been implemented in MATLAB, and the results have been evaluated. Data were collected from various datasets and pre-processed for each machine learning technique. A deep neural network was built with an input layer, hidden layers, and an output. Optimisation of the cascading classifier was mainly obtained by optimising each machine learning algorithm used and by starting using the parameters that gave the best results for each machine learning algorithm. The cascading classifier results show that the proposed methodology predicts transmembrane beta-barrel proteins topologies with high accuracy for randomly selected proteins. Using the cascading classifier approach, the best overall accuracy is 76.3%, with a precision of 0.831 and recall or probability of detection of 0.799 for TMB topology prediction. The accuracy of 76.3% is achieved using a two-layers cascading classifier. By constructing and using various machine-learning frameworks, systems were developed to analyse the TMB topologies with significant robustness. We have presented several experimental findings that may be useful for future research. Using the cascading classifier, we used a novel approach for the topology prediction of TMB proteins.

Author Related Publications

- Kazemian, H., Yusuf, S.A., White, K., and Grimaldi, C.M (2016), NN approach and its comparison with NN-SVM to beta-barrel prediction, *Expert Systems with* Applications 61:203–214
- Kazemian, H. and Grimaldi, C.M. (2020), Cascading classifier application for topology prediction of transmembrane beta-barrel proteins, *Journal of Bioinformatics and Computational Biology*, Vol.18, No. 3

Table of Contents

Acknowledgement 1

Abstract 2

Author Related Publications 4

1. Introduction 15

1.1 Motivation 16

1.2 Research Hypotheses 17

1.3 Aim and Objectives 18

1.4 Research methodology and design 19

1.4.1 Research philosophy 20

1.4.2 Exploratory Method 20

1.4.2.1 TMB datasets 21

1.4.2.2 Machine learning models 21

1.4.3 Constructive Method 21

1.4.3.1 Machine learning models development 21

1.4.3.2 Machine learning models optimisation 22

1.4.4 Experimental Method 22

1.5 Thesis structure 22

2. Literature Review 24

2.1 History and development of prediction methods 24

2.1.1 Discrimination of transmembrane strand proteins 24

2.1.1.1 Hidden Markov Models 25

2.1.1.2 Radial basis function networks with PSSM profiles 26

2.1.1.3 Nearest neighbor algorithm 28

2.1.2 Identification of membrane-spanning β-strand segments 29

2.1.2.1 Turn elimination method 29

2.1.2.2 Hydrophobicity profiles 29

2.1.2.3 Rule-based prediction 30

2.1.2.4 Artificial neural networks 30

2.1.2.5 Combination of methods 31

2.1.2.6 Recursive neural networks 32

2.2 Literature review on BOCTOPUS methods 33

2.2.1 BOCTOPUS1 method 33
2.2.2 BOCTOPUS2 method 34
2.3 Summary..34
3. Machine learning algorithms...36
3.1 Neural Networks...36
3.1.1 Artificial neural networks...36
3.1.1.1 Regularisation..38
3.1.1.2 Gradient-based optimisation..40
3.1.2 Recurrent neural networks...41
3.1.3 Convolution neural networks..43
3.1.4 Applications..44
3.2 K-Nearest Neighbors..45
3.2.1 General concept..45
3.2.2 Distance function..46
3.2.3 Advantages and disadvantages..47
3.3 Support Vector Machines...48
3.3.1 History of SVMs...48
3.3.2 SVM concept..48
3.3.2.1 Support Vector Machines: The linearly separable case..49
3.3.2.2 Support Vector Machines: The linearly non-separable case..50
3.3.2.3 Nonlinear Support Vector Machines..50
3.3.2.4 Quadratic programming problem of SVMs..51
3.3.2.5 Multi-class SVMs: One to others..53
3.3.2.6 Multi-class SVMs: Pairwise SVMs..54
3.3.3 Applications of Support Vector Machines...55
3.3.4 Limitations of Support Vector Machines...58
3.4 Ensemble methods..58
3.4.1 General concept..59
3.4.2 Types of ensembles...60
3.4.2.1 Voting and averaging based ensemble methods...60
3.4.2.2 Bootstrap aggregating (bagging)...60
3.4.2.3 Boosting..61

3.4.2.4 Multi-stage method cascading 61
3.4.2.5 Stacking 63
3.4.3 Applications of ensembles methods 64
3.5 Summary....68
4. Datasets and data preparation....70
4.1 Existing datasets 70
4.2 BOCTOPUS2 dataset 74
4.3 TOPDB dataset....77
4.4 Data preparation 80
4.4.1 Data collection....80
4.4.2 Data preprocessing 84
4.4.2.1 Boctopus2 structure array....84
4.4.2.2 Topbp structure array....85
4.4.3 Data transformation....87
4.4.3.1 Binarisation of the inputs and targets 87
4.4.3.2 Input and target matrices construction....89
4.5 Summary....90
5. Implementation of ANN, KNN, and SVM 91
5.1 Implementation of ANN....91
5.1.1 Model building and training 91
5.1.2 Hyperparameters tuning 95
5.1.2.1 Hidden layer 95
5.1.2.2 Transfer function 96
5.1.2.3 Training algorithms 97
5.1.2.4 Data division....99
5.1.3 Performance evaluation 105
5.1.3.1 Confusion matrix 105
5.1.3.2 Receiver operating characteristic curve....107
5.1.3.3 Prediction quality indices 109
5.2 Implementation of KNN....110
5.2.1 Model building and training 110
5.2.2 Hyperparameters tuning 113

5.2.3 Performance evaluation ... 115
5.2.3.1 Confusion matrix ... 115
5.2.3.2 Receiver operating characteristic curve ... 116
5.2.3.3 Prediction quality indices ... 117
5.3 Implementation of SVM ... 119
5.3.1 Model building and training ... 119
5.3.2 Hyperparameters tuning ... 126
5.3.3 Performance evaluation ... 134
5.3.3.1 Confusion matrix ... 134
5.3.3.2 Receiver operating characteristic curve ... 135
5.3.3.3 Prediction quality indices ... 136
5.4 Comparison of results ... 137
5.5 Summary ... 138
6. Implementation of cascading classifier ... 140
6.1 Model building and training ... 140
6.2 Hyperparameters tuning ... 142
6.2.1 Performance evaluation ... 142
6.2.1.1 Confusion matrix ... 146
6.2.1.2 Receiver operating characteristic curve ... 147
6.2.1.3 Prediction quality indices ... 148
6.3 Summary ... 149
7. Discussion of findings ... 151
7.1 Implications ... 151
7.2 Limitations ... 156
8. Conclusion and future research directions ... 159
References ... 164
Appendix A ... 191
Appendix B ... 192
Appendix C ... 193
Appendix D ... 194

List of Tables

Table 1 BOCTOPUS2 dataset 76

Table 2 Sample of 15 TMBs selected out of 123 TMBs available in TOPDB database..........79

Table 3 Best classification rate and area under the ROC curve on different binary encoding 88

Table 4 Training algorithms used 98

Table 5 Supported Kernel function names and their functional forms..................................125

List of Figures

Figure 1 Using PSSM profiles in radial basis function network 28
Figure 2 OmpA-Stretch of predicted amino acids residues in membrane-spanning beta-strands using TMBETA-NET 31
Figure 3 OmpA-Stretch of predicted BOCTOPUS pipeline 33
Figure 4 Schematic diagram of backpropagation training algorithm and typical neuron model 36
Figure 5 Gradient descent 40
Figure 6 Recurrent neural networks structure 42
Figure 7 AlexNet CNN architecture 44
Figure 8 Example of k-nearest neighbors classifier 46
Figure 9 Linear separating hyperplane: Separable case 49
Figure 10 Linear separating hyperplane: Non-separable case 50
Figure 11 Input space and feature space 51
Figure 12 Multi-class SVMs: Tree structure 54
Figure 13 Three methods for learning from heterogeneous data using SVM 57
Figure 14 Architecture of Multi-stage Deep Classifier Cascades and model details of (a) root node and (b) leaf nodes 62
Figure 15 Stacking 63
Figure 16 Schematic illustration of an ensemble using different classification 67
Figure 17 RCSB-PDB hierarchical classification of protein domain structures 73
Figure 18 BOCTOPUS 2: Improved topology prediction of transmembrane beta-barrel proteins 74
Figure 19 boctopus2_dataset_sequenceannotation.txt 75
Figure 20 Topology Data Bank of Transmembrane Proteins Topology, Structure and Prediction 77
Figure 21 Top_bp.txt 78
Figure 22 boctopus2Sequence.txt 81
Figure 23 boctopus2Labels.txt 82

Figure 24 TOPBPSequence.txt 83
Figure 25 TOPBPLabels.txt 83
Figure 26 boctopus2dataset.m 84
Figure 27 topbpdataset. mat. 85
Figure 28 Comput file 86
Figure 29 BinarizeInputs.m 87
Figure 30 BinarizeTargets.m 89
Figure 31 Construct input and target matrices 90
Figure 32 Neural network training GUI 92
Figure 33 Function `trainWithNeural` 93
Figure 34 Function `predictWithNeural` 93
Figure 35 Function `assessPerformance` 94
Figure 36 Representation of the neural network 95
Figure 37 Log-sigmoid transfer function 96
Figure 38 Tan-sigmoid transfer function 97
Figure 39 AUC & Predictive Accuracy Value 99
Figure 40 Structural assignment in the training data set 101
Figure 41 Structural assignment in the validation data set 102
Figure 42 Structural assignment in the testing data set 103
Figure 43 net.trainParam function results 104
Figure 44 Gradient 105
Figure 45 validation checks 105
Figure 46 Confusion matrix 107
Figure 47 Receiver Operating Characteristics curve 108
Figure 48 Quality indices of prediction 109
Figure 49 Function `trainWithKNN` 111
Figure 50 Function `predictWithKNN` 111
Figure 51 Scheme of a KD-Tree Search algorithm 112
Figure 52 Confusion matrix 116
Figure 53 ROC curve 117

Figure 54 Quality indices..118
Figure 55 Display of t from the command window...120
Figure 56 function `trainWithSVM`...121
Figure 57 function `predictWithKNN`..121
Figure 58 ECOC designs 123
Figure 59 Diagnostic message and convergence criteria...131
Figure 60 Illustration of the mapping φ 132
Figure 61 Confusion matrix for SVM classifier...135
Figure 62 ROC curve for SVM classifier..136
Figure 63 Quality indices of prediction for SVM classifier 137
Figure 64 Part of UseClassifiers.m for cascading classifier 140
Figure 65 Useclassifiers.m...141
Figure 66 Creating and training the SVM classifier at layer 2 142
Figure 67 Magnitude of gradient 144
Figure 68 Validation checks 144
Figure 69 Topology assignments in the training data set 145
Figure 70 Topology assignments in the validation data set...145
Figure 71 Topology assignments in the testing data set...146
Figure 72 Confusion Matrix 147
Figure 73 ROC curve..148
Figure 74 Quality indices performance 149

List of Acronyms

BAM	Binary Alignment/Map
BED	Browser Extensible Data
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
DAS	Dense Alignment Surface
DLDA	Diagonal linear Discriminant Analysis
ECOC	Error-Correcting Output Code
ERM	Empirical Risk Minimization
FSVM	Fuzzy SVM
GTF	Gene Transfer Format
HMM	Hidden-Markov Models
ISDA	Iterative Single Data Algorithm
KKT	Karush-Kuhn-Tucker
KNN	k-nearest neighbor
LS-SVM	Least Squares SVM
MAE	Mean Absolute Error
MATLAB	Matrix Laboratory
MEMSAT	MEMbrane protein Structure And Topology
NLP	Natural Language Processing
OAA-SVM	One-Against-All SVM
OMPs	Outer Membrane Proteins
PAM	Point Accepted Mutation
PDB	Protein Data Bank
PSSM	Position-Specific Scoring Matrix
RBF	Radial Basis Function
RNNs	Recurrent Neural Networks
ROC	Receiver operating characteristic
RVM	Relevance vector machine
SAM	Sequence Alignment/Map
SMO	Sequential Minimal Optimization
SNP	Single nucleotide polymorphism
SRM	Structural Risk Minimization
SVM	Support Vector Machines
TMB	Transmembrane beta-barrel
TMHMM	Transmembrane Helices; Hidden Markov Model
TOPDB	Topology Data Bank of Transmembrane Proteins

VC dimension	Vapnik Chervonenkis dimension
VCF	Variant Call Format
WIG	Wiggle

1. Introduction

Proteins are macromolecules or large biological molecules that consist of one or more long chains of amino residues. Protein secondary structure is the specific geometric shape caused by inter-molecular and intra-molecular hydrogen bonding of amide groups (Elmhurst College, 2003). There's been much research on alpha-helix membrane proteins. There are a few reasons for this. Alpha-helices contain an easily recognisable pattern of highly hydrophobic consecutive sequences, and rules such as the 'positive-inside rule' (von Heijne, 1992) can be applied. The positive-inside rule is a simple rule that defines how proteins insert and orient in membranes. Positive charges stay in the cytoplasm. Another reason, as explained earlier, is the profusion of alpha-helical membranes proteins (compared to beta-barrel transmembrane proteins) in entire genomes, and the same applies to datasets of proteins with experimentally solved structures in 3D.

There are significant improvements in the methods used to determine protein structures. The number of proteins with known structures at the atomic level still corresponds to a minimal fraction of known sequences (around 1%). There has been progress in technology used to understand the functions and structures of membrane proteins. For TMB proteins topology prediction, various techniques have been used. However, some areas are not fully explored. For example, the use of k-nearest neighbors for TMB topology prediction has not yet been implemented and evaluated. A complete comparison between k-nearest neighbors classifiers, support vector machine and deep neural network has not been evaluated. In a recent paper, Heffernan et al. (Heffernan, Paliwal, Lyons, Dehzangi, Sharma, Wang, Sattar, Yang and

Zhou, 2015) obtained an 82% prediction accuracy using a deep learning neural network. Recurrent neural networks provide successful results when applied to secondary structure prediction (Pollastri, Przybylski, Rost, Baldi, 2002), (Daniel, 2003). The use of Deep learning towards transmembrane beta-barrel topology prediction could provide a significant advance in this field.

1.1 Motivation

The prediction of secondary structure helps to define the 3D structure of proteins, but 100% single residue accuracy is not the goal. There are a few reasons why the accuracy of 100% has not been achieved so far. There are uncertainties and errors in the Protein Data Bank (PDB), and some regions are classified as disordered instead of ordered in the PDB. This leads to noisy data. Also, prediction errors can originate from a rigid secondary structure definition without considering differences (Magnan and Baldi, 2014).

One of the most critical goals in bioinformatics is accurately predicting protein secondary structures. It is essential in the design of drugs or novel enzymes. The location of beta-barrel transmembrane proteins is in the outer membranes of gram-negative bacteria, outer membranes of mitochondria or chloroplasts. They have critical functions such as passive nutrient intake ion-transport. Accurate predictions of secondary and tertiary structures of transmembrane proteins are therefore needed. Effective antibacterial drugs are developed with a good knowledge of the 3D structures of transmembrane beta-barrels.

Efforts related to beta-barrel topology prediction have been overshadowed, and there could be an improvement in prediction accuracy. Recent studies focus on alpha-helix transmembrane regions prediction using an SVM- genetic algorithm (Kazemian et al., 2013) or adaptive neural fuzzy inference system (Kazemian et al., 2014), for example. A recent paper embarks upon a NN (Neural Network) technique and its comparison with hybrid-two-level NN-SVM (Support Vector Machines) methodology to classify inter-class and intra-class transitions to predict the number and range of beta membrane-spanning regions (Kazemian et al., 2016). The computer simulation results demonstrate a significant impact and a superior performance of NN-SVM tests with a five residue overlap for signal protein over NN with and without redundant proteins for predicting transmembrane beta-barrel spanning regions. Recent studies focus on alpha-helix transmembrane regions prediction with the use of SVM- genetic algorithm (Kazemian et al., 2013) or adaptive neural fuzzy inference system (Kazemian et al., 2014). This PhD thesis aims to evaluate what prediction methods and techniques have been used so far and find new approaches and strategies that would improve the prediction accuracy results obtained from this implementation could represent an essential advancement in the prediction of the topology of beta-barrel transmembrane proteins with the use of a computer simulation. Deep learning is a rapidly evolving field, and a significant advance in bioinformatics could be provided while using those algorithms as part of the thesis. The most common deep learning architectures are convolutional deep neural networks. Transmembrane beta-barrel proteins datasets are usually small. The performance of multiple algorithms was evaluated in a recent article (Sharma et al., 2016) using small datasets with various dimensionalities. Their report indicated that KNN (k-nearest neighbors), SVM (Support vector machines) and linear discriminant is the best algorithms when using small datasets. Some of those models will be implemented as part of the thesis. The PhD thesis will include a comprehensive evaluation and comparison of the various machine learning applications. The results obtained from this paper will be applied and utilised to develop and improve a complete scientific website referenced as the 'transmembrane and signal peptide topology prediction web server'. The objective of this server is to contribute to knowledge and understanding in the field of transmembrane proteins, signal peptides and machine learning algorithms through the development of an integrated software suite. The server includes already an ensemble of artificial intelligence techniques such as artificial neural networks, fuzzy inference systems, genetic algorithms, hidden Markov models and support vector machines. Predicting beta-barrels, alpha-helices, and signal peptides in one website is unprecedented, and the site will compete with the only few world-renowned websites, such as TMHMM 2.0 (Krogh, Larsson, von Heijne & Sonnhammer, 2001), DAS (Cserzo, Wallin, Simon, von Heijne and Elofsson, 1997), MEMSAT (Jones, Taylor & Thornton, 1994) and SOSUI (Hirokawa, Boon-Chieng and Mitaku, 1998).

1.2 Research Hypotheses

The key research questions, which are the basis of this research, are:

1. Which machine learning technique is the most appropriate and has the best predictive accuracy for the application of TMB topology prediction?
2. What model could be built to improve current predictions models available today?

This study will therefore need to address the following research questions:

- TMB topology prediction is a saturated research area, and current models available have low accuracy. What factors limit the improvement of prediction accuracy?

- How to develop a novel model that can combine multiple machine learning techniques? What architecture is the most appropriate?

- How can the model be optimised to improve TMB topology prediction accuracy?

- Can the proposed model add a significant contribution of knowledge to the topic and be helpful in fields such as medicine?

1.3 Aim and Objectives

This research aims to evaluate the performance of selected machine learning techniques in predicting TMB topologies and compare them. The goal of the prediction is to provide the topology of beta-barrel within a protein or protein family and more specifically, transmembrane proteins. The thesis aims at generating new knowledge in the field of bioinformatics. Proteins are vital parts of many biological processes. Prediction methods are of great importance for membrane proteins and are very helpful in drug discovery. If there is a need to design new drugs, cure diseases, then it is necessary to understand the actual molecular structure that drugs bind to and then better and improved drugs can be developed. It is hoped that the research project will develop new understandings in the field of machine learning.

The main objectives are:

1. Review the various types of machine learning techniques and existing algorithms that could be applied for beta-barrel TM topology prediction.
2. Review the various types of proteins, protein structures, and general aspects.
3. Use an appropriate methodology to collect, prepare and encode data using various datasets.
4. Develop ANN, KNN, SVM and cascading classifier models.
5. Evaluate the performances of the proposed methods, analyse, and compare the results.

1.4 Research methodology and design

This chapter examines the research methodology adopted in this thesis. It first defines the philosophy that supports the approach taken with the research, discussing the use of a positivism posture to study and the consequent choice of the exploratory, constructive, and experimental approaches used. Hussey et al. (Hussey & Hussey, 1997) discuss methodology and method. The writers define the methodology as the overall approach to the research process encompassing a body of methods and define a method as the various techniques of collecting and/or analysing data. Mason (2002) describes the concept of methodological strategy, indicating that a particular method can be part of the strategy. The approach here includes all aspects of the research process under the overall methodology. The research design and the methods used, the data collection method chosen, and the means of analysis are all considered part of the methodology and are defined in the following sections.

The methodology combines exploratory, constructive, and experimental methods. The exploratory method aims to gain insights and familiarity with the topic of TMB topology prediction. Multiple methods have been used to effectively conduct the proposed research and test the model sets and machine learning techniques used. To ensure good research design, it is also necessary to evaluate the data sources to make sure that they will help answer research questions most effectively. The constructive method is used for building the models. The experimental method is used for testing the models and tuning the hyperparameters.

1.4.1 Research philosophy

A positivism approach was adopted in this research. As a philosophy, positivism adheres to the view that only knowledge based on facts gained through measurements, for example, is trustworthy. In positivism studies, the role of the researcher is limited to data collection and interpretation objectively. In studies such as this thesis, research findings are observable and quantifiable. There is often a distinction regarding research philosophies between positivism and interpretivism (Bryman & Bell, 2007). In positivism, the purpose of research is a scientific explanation. Researchers who work from this perspective explains in quantitative terms how variables interact, shape events, and cause outcomes. They often develop and test these explanations in experimental studies. This framework maintains that reliable knowledge is based on direct observation or manipulation of natural phenomena through empirical and experimental means (Lincoln & Guba, 2000; Neuman, 2003).

1.4.2 Exploratory Method

The exploratory method aims to look for patterns, ideas, or hypotheses. It helps determine the best research design, data collection method, selection of datasets and machine learning techniques. It relies on secondary research, such as reviewing available literature and datasets used.

1.4.2.1 TMB datasets

A high-quality data set is used for training and validation purposes when constructing any prediction. An extensive literature review is performed on the list of datasets used in recent models and the list of available datasets that are available. Errors in databases are not infrequent, and adding them is an element of noise. While such noise is often well tolerated by machine learning, the problem is more significant in smaller data sets such as the one used for TMB topology prediction.

1.4.2.2 Machine learning models

The initial literature review is used to find the various machine learning approaches used to predict beta-barrel membrane proteins topology. Machine learning approaches prevail over hydrophobicity methods due to their statistical formulation. There are few machine learning-based beta-barrel TM topology predictors available based on ANN, HMM, SVM or a combination of those machine learning techniques. An extensive literature review is performed on the accuracy of the various methods described in various journals and articles.

1.4.3 Constructive Method

This computing environment and computing language are selected. Models are implemented in MATLAB. The models are continually learning and adapting to new data. Hyperparameters for all algorithms are modified.

1.4.3.1 Machine learning models development

This activity identifies the various functions in MATLAB used to create the models. It defines the encoding techniques that are used. The architectures of the models are described and implemented.

1.4.3.2 Machine learning models optimisation

This activity has the goal of improving performance. It assesses which model is the most accurate for TMB topology prediction. Hyperparameters for ANN, KNN, SVM and cascading classifier are modified.

1.4.4 Experimental Method

This method proves that the models created generate a better TMB topology prediction accuracy. The data are collected from various online transmembrane beta-barrel datasets. The activities include testing the models. Multiple runs are executed, and results are provided. The analysis of results is evaluated with the use of performance graphs.

1.5 Thesis structure

To report the findings of the research in detail, the remainder of the thesis is organised as follows:

- Chapter 2 presents the literature review. Multiple research papers have been evaluated and analysed in the context of TMB topology prediction. The available current tools, existing research, methodologies, and limitations have been described in this chapter.

- Chapter 3 provides some theories and concepts related to machine learning algorithms such as SVM, ANN, KNN and ensemble methods.

- Chapter 4 presents the various datasets of transmembrane proteins that are available and the techniques used for data preparation.

- Chapter 5 presents the implementation of ANN, KNN and SVM models in MATLAB. Modification to the hyperparameters has been described. Results of the implementations have been provided.

- Chapter 6 presents the implementation of a cascading classifier in MATLAB. Modification to the hyperparameters has been described. Results of the implementations and runs have been provided.

- Chapter 7 provides the discussion of findings.

- Chapter 8 provides the conclusion of the thesis, describes the achievements and contributions, and discusses future research.

2. Literature Review

This chapter presents a review of publications in areas of transmembrane strand proteins. It gives an overview of what has been achieved and used methods. Progress has been shown for the past several decades, and still, the accuracy is modest

2.1 History and development of prediction methods

Three types of computational problems are related to transmembrane beta-barrel. Transmembrane beta-barrel detection and discrimination (from other proteins) is the first type. Transmembrane topology prediction and transmembrane beta-contacts prediction are the second and third types. Machine learning can be applied to all these activities.

In the 1950s, Pauling et al. (Pauling and Corey, 1951) looked at creating alpha-helix and beta-strand local conformations. Chou and Fasman developed the Chou-Fasman method in 1974 described in two different papers (Chou and Fasman, 1974a) and (Chou and Fasman, 1974b). Early prediction methods were developed using a simple analysis of how amino acids are distributed in beta-strands (and alpha-helices). Chou et al. statistical method is based on amino acids propensities defined as natural inclinations or tendencies to belong to a given secondary structure. The propensity of a position is calculated using an average of 5 residues (for a strand) surrounding each position rather than a position-by-position analysis. However, this method is limited due to low accuracy, unreliable parameters, and over prediction. Some methods used for the discrimination of transmembrane strand proteins and identification of membrane-spanning β-strand segments are described in the following paragraphs.

2.1.1 Discrimination of transmembrane strand proteins

Outer membrane proteins (OMPs), also known as a β-barrel membrane or transmembrane strand proteins, perform various functions, such as mediating nonspecific, passive transport of ions and small molecules, selectively allowing the passage of molecules such as maltose and sucrose. The success rate of discriminating β-barrel membrane proteins from other proteins is significantly lower than that of α-helical membrane proteins. Multiple methods have been proposed for determining OMPs. These methods are based on hydrophobicity, sequence alignment, neural networks, HMMs, conformational parameters, statistical methods, nearest neighbor algorithms, SVMs, and machine learning techniques.

2.1.1.1 Hidden Markov Models

HMM-B2TMR is based on HMM to predict beta-barrel transmembrane proteins topology (Martelli, Fariselli, Krogh and Casadio, 2002). The novelty in this method is the development of a specific input that is based on multiple sequence alignment. The prediction accuracy is 83% in a jackknife test. A non-redundant dataset of 12 OMPs is used for training and testing. Fariselli et al. developed a later version. They introduced a decoding algorithm called the posterior-Viterbi algorithm (Fariselli, Martelli and Casadio, 2005). A decoding algorithm is needed when HMMs predict a given feature. Fariselli et al. used the previously HMM developed by Martelli et al. to test their decoding algorithm. Profiles from PSI-BLAST are used for inputs (Altschul, Madden, Schaffer, Zhang, Zhang, Miller and Lipman, 1997).

PRED-TMBB is a server based on HMM (Bagos, Liakopoulos, Spyropoulos and Hamodrakas, 2004a). A non-redundant dataset of 14 OMPs was used for the training. A training set that is constituted of 16 non-homologous OMPs (Bagos, Liakopoulos, Spyropoulos and Hamodrakas, 2004b) was used later in the model for retraining. The training follows the conditional maximum likelihood method. Single sequences are used as input. The Viterbi algorithm, N-best algorithm, or posterior decoding, in addition to the dynamic programming algorithm, are also used (Bagos, Liakopoulos, Spyropoulos and Hamodrakas, 2004a). The user can select one of the three different decoding options.

ProfTMB is based on HMM (Bigelow, Petrey, Liu, Przybylski, Rost, 2004). In the original paper, the impact of using different profiles on ProfTMB accuracy was not evaluated. The method described in the latest paper (Bigelow and Rost, 2006) reached an overall four-state accuracy as high as 86%. The algorithm is mainly based on HMM-B2TMR with some modifications, like having four states for each residue defined as up-strand, down-strand, periplasmic-loop, and outer-loop (Bigelow and Rost, 2006). The C++ source code can be downloaded and compiled by the users, and the original training data or a modification of it can be used with PROFtmb.

BETA-TM is a predictor based on HMM (Ahn, Yoo and Park, 2003). The Baum-Welch algorithm with a dataset of 11 non-homologous proteins is used for training. The Viterbi algorithm is used for decoding.

TMB-HMM is also an HMM-based topology predictor. The residues in the transmembrane can be predicted as exposed to the membrane versus hidden in the protein structure (Singh, N., Goodman, Walter, Helms, and Hayat, 2011). The residues that do not belong to beta-barrel strands but are in the transmembrane region can also be predicted. TMB-HMM uses frequency profiles obtained from MSAs as input. It has been trained on a dataset of 19 TMBs. It has a three-state prediction accuracy of 72%. The predictor was trained on a relatively small training dataset, but it is expected that the accuracy will improve once more 3D structures of TMBs become available.

2.1.1.2 Radial basis function networks with PSSM profiles

Radial basis function networks have been suggested to predict the number of beta-barrel strands and membrane-spanning regions in beta-barrel OMPs (Ou, Chen and Gromiha, 2010). In radial basis function networks, the hidden units perform the computation. To define the topology of beta-barrel OMPs, it is essential to predict accurately the number of beta-strands present in OMPs. This prediction is also important to determine the correct assignment of strands in the membrane. Ou et al. created a protocol to predict the number of beta-strand in OMPs. If there are more than 590 residues in the proteins, proteins will be predicted as having 22 beta-strands. If there are less than 200 residues in the proteins, proteins will be predicted as having 8 beta-strands. If the sequence length is 200500 residues, a radial basis function network is used simultaneously with amino acid compositions such as Alanine, Aspartic acid, histidine, Tyrosine, and Valine to predict

10,12,14,16 and 18 beta-strands, respectively (Ou, Chen and Gromiha, 2010). A cross-validation accuracy of 96.4% is achieved for the correct prediction on the beta-strands number. For the prediction of membrane-spanning regions, residues in transmembrane beta-strands are identified using a radial basis function network that has been trained with PSSM profiles (obtained from PSI-BLAST). With this method, there is a reduction in over-prediction and under-prediction.

Over-prediction and under-prediction are significant problems in transmembrane strands prediction. The method has an accuracy of 87%. Figure 1 represents the use of PSSM profiles as features in the radial basis function network as described in the Ou et al. paper. The PSSM profiles are generated from PSI-BLAST (Altschul, Madden, Schaffer, Zhang, Zhang, Miller and Lipman, 1997). A 15x20 matrix is generated using a window size of 15. It is used as an input for the radial basis function network.

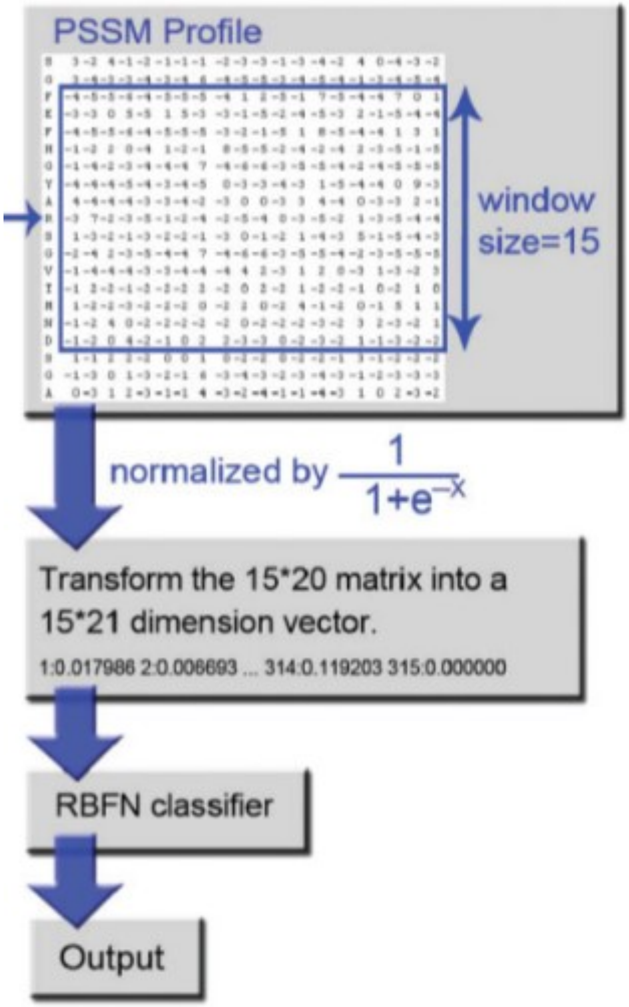


Figure 1 Using PSSM profiles in radial basis function network

Source: Ou, Y. Y., Chen, S. A and Gromiha, M. M. (2010)

2.1.1.3 Nearest neighbor algorithm

Amino acid composition is one of the parameters, which can be used to identify β-barrel membrane proteins (Gromiha, 2005). Garrow et al. (2005) used the amino acid co mposition and proposed a modified k-nearest neighbor algorithm, TMB-HUNT, to classify the proteins into transmembrane β-barrel (TMB) and non- TMB. This method showed an accuracy of 92.5% using weighted amino acids and evolutionary information.

2.1.2 Identification of membrane-spanning β-strand segments

2.1.2.1 Turn elimination method

Another approach used to predict and identify segments causing polypeptides to reverse their direction is called turn identification and developed by Paul and Rosenbusch (Paul and Rosenbusch, 1985). With the removal of beta-turns and the selection of a length of 6 residues for a strand, they predicted transmembrane beta-strands.

2.1.2.2 Hydrophobicity profiles

In biochemistry, an amphipathic molecule can interact on one side with nonpolar or hydrophobic molecules and on the other side with polar or hydrophilic molecules. Beta-sheets are often amphipathic and fold into the rest of the protein to protect the hydrophobic sidechains from water environments. Vogel and Jähnig developed a method for predicting transmembrane beta-strands, and the method is based on the amphipathic characteristic of beta-strands (Vogel and Jähnig, 1986). Jähnig (Jähnig, 1990) indicated that hydrophobicity analysis could be sufficient to predict amphiphilic alpha-helices and beta-strands proteins that cross both sides of a membrane (Gromiha, 2010). Welte et al. (Welte, Weiss, Nestel, Weckesser, Schiltz and Schulz, 1991) and Cowan et al. (Cowan and Schirmer, 1992) also proposed a method based on the physicochemical properties of amino acids for the prediction of transmembrane beta-barrel proteins. In 1993, Schirmer et al. applied the algorithm of Kyte and Doolittle (Kyte and Doolittle, 1982) to identify transmembrane proteins (Schirmer and Cowan, 1993). The Kyte and Doolittle algorithm (Kyte and Doolittle, 1982) determines the mean hydrophobicity within a sliding window. Gromiha et al. (Gromiha and Ponnuswamy, 1993) have used amino acid hydrophobic properties to predict beta-strands. The method introduces two characteristics: It is not dependent only on the amphipathic character of a sequence segment while identifying it as a transmembrane strand. The method can predict strands in varying lengths. This method has an accuracy of 76% for predicting transmembrane beta-strands when used with porin from Rhodobacter capsulatus.

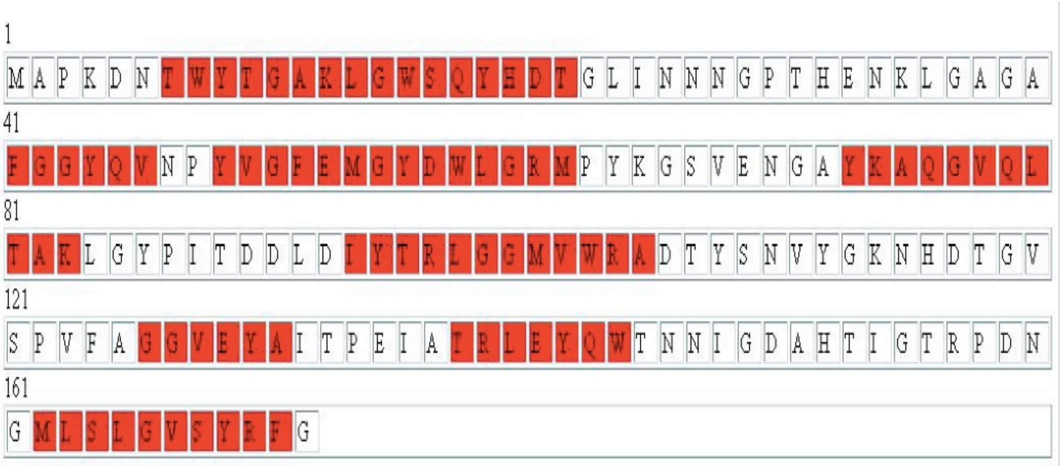
2.1.2.3 Rule-based prediction

Gromiha et al. (Gromiha, Majumdar and Ponnuswamy, 1997) suggest predicting transmembrane beta-strands using a rule-based approach. This is an important class of methods. The predictor will use the statistical properties of amino acids. There are primary rules and secondary rules. Rules consider hydrophobicity of amino acids or amphipathicity of beta-strand segments, for example. (Gromiha, Majumdar and Ponnuswamy, 1997). This method has a prediction accuracy of 82% for all the bacterial porins evaluated. A disadvantage of this approach is that training sets based on are limited. Also, when there is no similarity between the sequences and the proteins of the training set, it is more difficult to obtain the structural characteristics of the bacterial OMPs. This method is used as part of the software package named BioSuite (The NMITLI-BioSuite Team, 2007) used in bioinformatics.

2.1.2.4 Artificial neural networks

The first method, proposed by Diederichs et al., is using a neural network to predict the topology of bacterial OMPs. The neural network is like other neural networks used, such as Holley et al. (Holley and Karplus, 1998); however, they use a smaller input window and one output unit (Diederichs, Freigang, Umhau, Zeth and Breed, 1998). A dataset containing seven bacterial porins was first used for the training, but a new dataset included some new solved (non-porins) structures for retraining. The name of the server is OM_Topo_predict. The server was not available at the time this research was conducted.

The use of neural networks for the prediction of transmembrane beta-barrel in OMPs was suggested by Gromiha, Ahmad and Suwa. The method uses single sequence information as input. The neural network comprises three layers (Gromiha, Ahmad and Suwa, 2004). TMBETA-NET is a predictor for identifying beta-strand proteins that cross both sides of a membrane. It used a neural network. Empirical rules are included to remove not likely predictions of transmembrane strands, such as a strand with three residues (Gromiha, Ahmad and Suwa, 2005). The predictor is, therefore, more precise. It achieves a prediction accuracy of 73% for membrane-spanning beta-strands. Applications of TMBETA-NET are shown in figure 2. The figure represents the stretch of amino acid residues in the transmembrane strand for OmpA using an amino acid sequence as input. There is height membrane-spanning beta-strands segments in OmpA, as represented below.



```
1
M A P K D N T W Y T G A K L G W S Q Y H D T G L I N N N G P T H E N K L G A G A
41
F G G Y Q V N P Y V G F E M G Y D W L G R M P Y K G S V E N G A Y K A Q G V Q L
81
T A K L G Y P I T D D L D I Y T R L G G M V W R A D T Y S N V Y G K N H D T G V
121
S P V F A G G V E Y A I T P E I A T R L E Y Q W T N N I G D A H T I G T R P D N
161
G M L S L G V S Y R F G
```

Figure 2 OmpA-Stretch of predicted amino acids residues in membrane-spanning beta-strands using TMBETA-NET

Source: Gromiha, M.M., Ahmad, S. and Suwa, M. (2005)

2.1.2.5 Combination of methods

B2TMPRED is a predictor that combines dynamic programming (Jacoboni, Martelli, Fariselli, De Pinto and Casadio, 2001) and a neural network. Sequence profiles derived from PSI-BLAST (Altschul, Madden, Schaffer, Zhang, Zhang, Miller and Lipman, 1997) are used as inputs. The method is trained using a dataset of 11 OMPs. Dynamic programming is used to identify the location of the TM strands. The discrimination of membrane beta-strands from extramembrane regions is achieved by creating and training a feed-forward neural network.

Secondary structure prediction accuracy is as high as 78%. The predictor is available at: http://gpcr.biocomp.unibo.it/cgi/predictors/outer/pred_outercgi.cgi

TBBpred is a predictor that combines neural networks and support vector machines (Natt, Kaur, Raghava, 2004). The neural network part of the predictor uses evolutionary information derived from multiple alignments. The support vector machines module uses physicochemical properties. One of the methods alone can be selected by the user, but the authors indicate that combining the methods (81.8%) will significantly improve the prediction accuracy.

ConBBPred is a web server that combines individual predictors to a single consensus prediction. Bagos et al. compared the performance of different methods for predicting beta-barrel OMPs topology using a non-redundant dataset of 20 beta-barrel OMPs (Bagos, Liakopoulos and Hamodrakas, 2005). They indicated that methods based on HMM are the best predictors. When only transmembrane beta-barrel domains are used, predictors achieve better results. The consensus prediction method is also using a dynamic programming algorithm.

Zou et al. present a model that combines HMM and genetic algorithms (Zou, Wang, Wang, and Hu, 2010). When designing HMM-based methods, the algorithms used for parameter estimation and decoding are essential. The Baum-Welch algorithm is often selected for the training of HMM for the prediction of TMB. Zou et al. use a genetic algorithm for training and the posterior-Viterbi algorithm for decoding. The dataset includes 33 TMBs. It is one of the largest datasets used so far in the literature. Zou et al. indicate that their method achieves better results than all other methods for topology prediction.

2.1.2.6 Recursive neural networks

TMBpro suite is a three-stage method for transmembrane beta-barrel topology prediction, beta-contacts, and tertiary structure prediction. A 1D-recursive Neural Network (1D-RNN) and dynamic programming refinement are used (Randall, Cheng, Sweredoski and Baldi, 2008). Recurrent neural networks are recursive neural networks with a particular structure. Recursive neural networks operate on any hierarchical structure, while recurrent neural networks operate on the linear progression of time. According to Q2 (84.2%) and MCC (0.720), TMBproSS outperforms PRED-TMBB. TMBpro-SS is a secondary structure predictor, TMBpro-CON is a beta-contacts predictor, and TMBpro-3D is a tertiary structure predictor.

2.2 Literature review on BOCTOPUS methods

2.2.1 BOCTOPUS1 method

BOCTOPUS is a recent method that has been used for the prediction of transmembrane beta-barrel topologies is (Hayat and Elofsson, 2012). Support vector machines have been utilised to predict the local structural preferences for a residue. An HMM model has been used for the topology model for proteins. The architecture of BOCTOPUS is represented in figure 3.

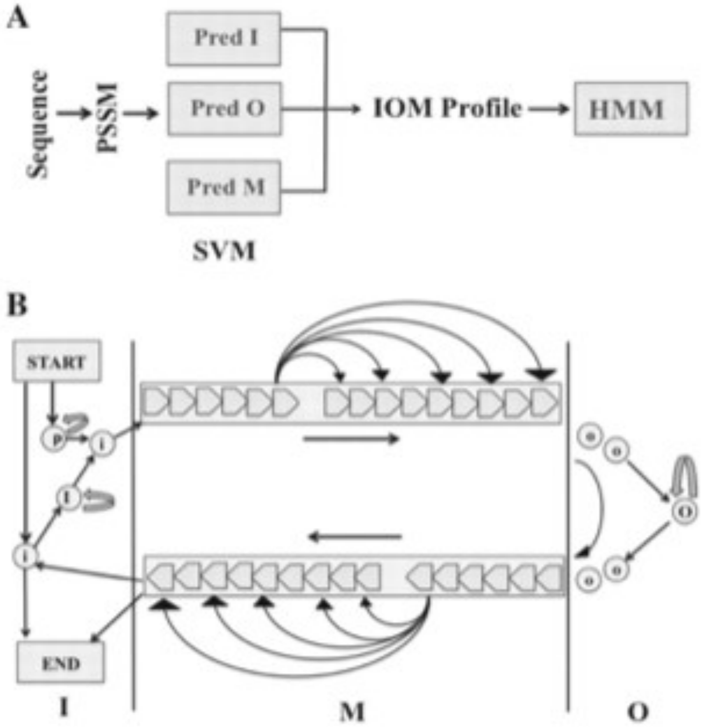


Figure 3 OmpA-Stretch of predicted BOCTOPUS pipeline

Source: Hayat, S. and Elofsson, A. (2012)

PSI-BLAST provides the PSSM for a given sequence (Altschul, Madden, Schaffer, Zhang, Zhang, Miller and Lipman, 1997). The prediction of the residue-level preference for each amino acid is obtained with three SVMs. The preference can be I, M or O. HMM uses an 'IOM' profile acquired to predict the topology. The Viterbi algorithm is used to obtain the final topology. I correspond to the inner loop, O outer loop and M transmembrane beta-strand.

2.2.2 BOCTOPUS2 method

Hayat et al. developed BOCTOPUS2 in 2016. It is an updated version of BOCTOPUS (Hayat, Peters, Shu, Tsirigos and Elofsson, 2016). This method can identify Barrel domains and topologies and predict residues' orientation in transmembrane beta-strands (Hayat, Peters, Shu, Tsirigos and Elofsson, 2016). The prediction accuracy is 69%. It is an increase of 10% in comparison to BOCTOPUS. BOCTOPUS2 is trained on 42 full Uniprot sequences with a known 3D structure. It consists of two stages. The prediction of per-residue location (referred to as inner-loop, outer-loop, membrane lipid-facing and membrane pore-facing) is achieved using four SVMs in the first stage, and an HMM predicts the overall topology in the second stage (Hayat, Peters, Shu, Tsirigos and Elofsson, 2016). Regarding the k-nearest neighbors algorithm, it has not been used for the prediction of TMB topologies prediction. TMB-Hunt (Garrow, Agnew and Westhead, 2005) is a web server that uses k-nearest neighbors only to discriminate between non-TMB proteins and TMB proteins based on the composition of amino acids. This is a different problem than predicting TMB topologies.

2.3 Summary

The literature review critically analysed published academic literature, mainly peer-reviewed papers and books, on TMB topology prediction. It was possible to present an overview of the current knowledge gained from previous work. When discussing each relevant piece of literature, the review highlighted the gaps and strengths and weaknesses of a particular study, paper, or book. The review of the literature review helps to have a clear understanding of what has been published in the subject area of research. Critical theories in the field of TMB topology prediction have been evaluated. Leading research groups and authors in the field and their essential contributions to the research topic have been identified. A clear understanding of the research gap helped redefine the PhD research questions.

3. Machine learning algorithms

This chapter provides a description and overview of the three essential types of neural networks that form the basis for most models in deep learning. K-nearest neigbhors algorithms, Support Vector machines and ensemble methods. Some general guidelines for the practical methodology involved in designing, building, training, and configuring a deep neural network will be presented and some of the current applications of deep learning.

3.1 Neural Networks

3.1.1 Artificial neural networks

Artificial Neural Network, or ANN, is a group of multiple neurons at each layer. ANN is also known as a Feed-Forward Neural network because inputs are processed only in the forward direction. Figure 4 represents a schematic diagram of a backpropagation training algorithm and a typical neuron model.

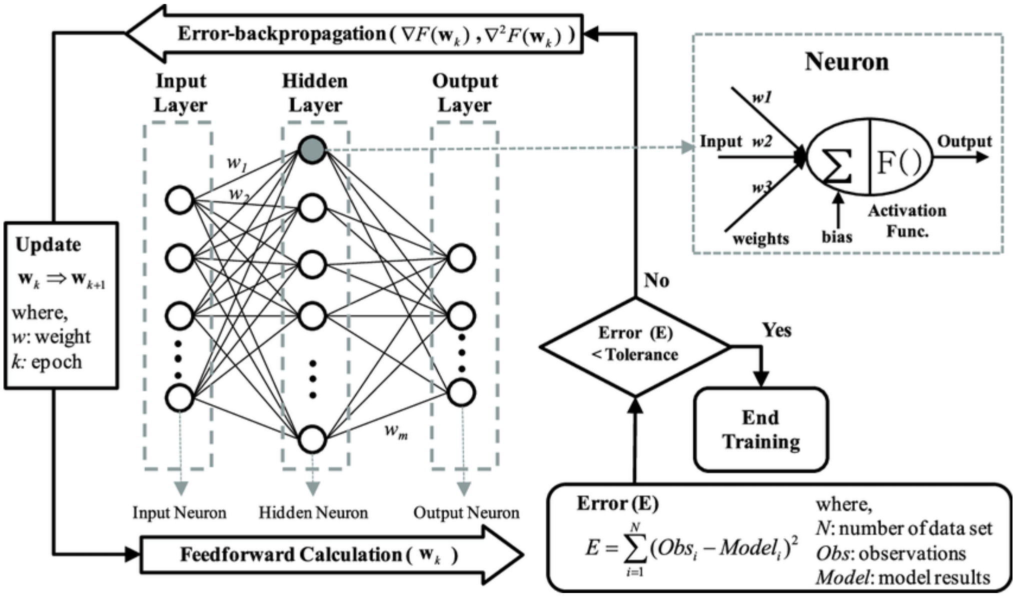


Figure 4 Schematic diagram of backpropagation training algorithm and typical neuron model

Source: Kim, S.E. and Seo, I.W. (2015)

ANN consists of 3 layers (Input, Hidden and Output). The input layer accepts the inputs, the hidden layer processes the inputs, and the output layer produces the result. Essentially, each layer tries to learn certain weights. A single perceptron (or neuron) can be imagined as a Logistic Regression. ANN can solve tabular data, image data, or text data problems.

Artificial Neural networks can learn any nonlinear function. Hence, these networks are popularly known as Universal Function Approximators. ANNs can learn weights that map any input to the output.

One of the main reasons behind universal approximation is the activation function. Activation functions introduce nonlinear properties to the network. This helps the network learn any complex relationship between input and output. One common problem in all these neural networks is the Vanishing and Exploding Gradient. This problem is associated with the backpropagation algorithm. The weights of a neural network are updated through this backpropagation algorithm by finding the gradients. So, in the case of a very deep neural network (network with many hidden layers), the gradient vanishes or explodes as it propagates backwards, which leads to vanishing and exploding gradient.

A recent article (LeCun, Bengio and Hinton, 2015) indicates that deep neural networks outperform conventional methods in the areas of speech recognition or visual object detection. The ideal features for protein structure prediction problems have not been identified yet. DNSS is a predictor for secondary structure using deeper neural networks (Spencer, Eickholt, and Cheng, 2015). The prediction accuracy is 80.7%. It is essential to indicate that this predictor and other papers referring to predictors using deep neural networks refer to TMB topology prediction.

Feed-forward neural networks are called feedforward as the information flows through the function that is evaluated from x, through the intermediate computations used to define the function and finally to the output y. This is called forward propagation. There are no feedback connections in which outputs of the model are fed back into itself. When feedback connections are included in feedforward networks, they are called recurrent neural networks, discussed later in the paragraph. Feedforward neural networks are essential to machine learning practitioners, and they are used in many commercial applications. Feedforward networks are called networks because they are usually represented by many different functions. These chain structures are the most often used structures of neural networks. The overall length of the chain gives the depth of the model. From this terminology, the term ‘deep learning’ comes from. Hinton et al. define a deep neural network as a feed-forward artificial neural network with more than one layer of hidden units between the inputs and outputs (Hinton, Deng, Yu, Dahl, Mohamed, Jaitly, Senior et al., 2012). The learning algorithm must decide how to use those layers to produce the desired output, but the training data does not tell what each layer should do. Finally, these networks are called neural as they are derived from neuroscience models. Each unit in hidden layers is like a neuron as it receives an input from many other units and computes its activation value. Feed-forward networks are not models of brain function but are designed to achieve statistical generalization. Deep feedforward neural networks have full potential and can be applied to various tasks. Advancements in optimisation to improve the algorithms and model design are expected to perform further.

3.1.1.1 Regularisation

Regularisation refers to any modification that can be done to a learning algorithm that will reduce the generalisation error. The training error will not be reduced. Regularisation strategies for deep models will be covered in this paragraph. Those strategies can also be used for models that can be utilised as a part of deep learning models. Dataset augmentation is one strategy that can be used. If machine learning is trained on more data, it will generalise better. In practical cases, there is a limitation on the amount of data, especially in the situation of beta-barrel proteins. One way to circumvent this problem has been the creation of artificial data and adding this data to the training set. This approach is notably easier for classification problems such as object recognition and efficient for speech recognition problems (Jaitly and Hinton, 2013). It is not readily applicable to other tasks such as pattern recognition. Data augmentation also includes noise in the input (Sietsma and Dow, 1991); however, deep neural networks are not effective when noise is added. Improvement of the robustness of deep neural networks can be performed by using random noise applied to their inputs during training (Tang and Eliasmith, 2010). If the magnitude of the noise is meticulously tuned, the process can be very efficient (Poole et al., 2014). Therefore, it is essential to consider if the dataset augmentation has been applied when comparing machine learning benchmark results. Dataset augmentation constructed manually can often significantly reduce the generalisation error of machine learning techniques. If a comparison of the performance of one machine learning technique to another one is needed, the use of controlled experiments is recommended. Another strategy that has been used is to add noise to the weights as described in this paper (Jim et al., 1996; Graves, 2011) with recurrent neural networks. A recent article (Luo and Yang, 2014) studied the effect of introducing different noise into different components of different types of deep learning neural networks. They observed that a reasonable amount and a reasonable magnitude of noise could improve the model's accuracy and convergence rate when introduced into a deep learning model. The noise was added to the gradient descent component of Logistic Regression into weights between layers for Multi-layer Logistic Regression. During a noise-free training process of the model, weights between layers are transmitted and updated without any loss of information or variances. However, during a noise added training process, weights between layers are subject to some variation. Lastly, the noise was added into the feature mapping component of a Convolutional Neural Network.

Regularisation also includes the technique of early stopping. No changes are needed to the primary training process, the definition of the function, or the ensemble of parameter values that are allowed. It does not incur damages to the learning dynamics. It can be with other regularisation strategies and on its own. A validation set is required for early stopping. It signifies that some training data is not injected into the model. The computational cost of the training process is also reduced with early stopping. A generalisation can be reduced by combining multiple models (Breiman, 1994). This technique is called bagging (bootstrap aggregating), part of a large approach called model averaging. This technique is also referred to as the ensemble method. The different models will habitually not make all the same errors on the test set, which is why this model works. It involves constructing different datasets. It is extremely powerful for reducing generalisation errors.

3.1.1.2 Gradient-based optimisation

Optimisation consists of minimising or maximising a function by finding the best value of an argument. The optimisation is included in most deep learning algorithms. The objective function (or criterion) represents the function that needs to be minimised or maximised. When minimised, it is called the cost function (or loss function/error function). The term differs depending on machine learning publications. Figure 5. represents the gradient descent technique.

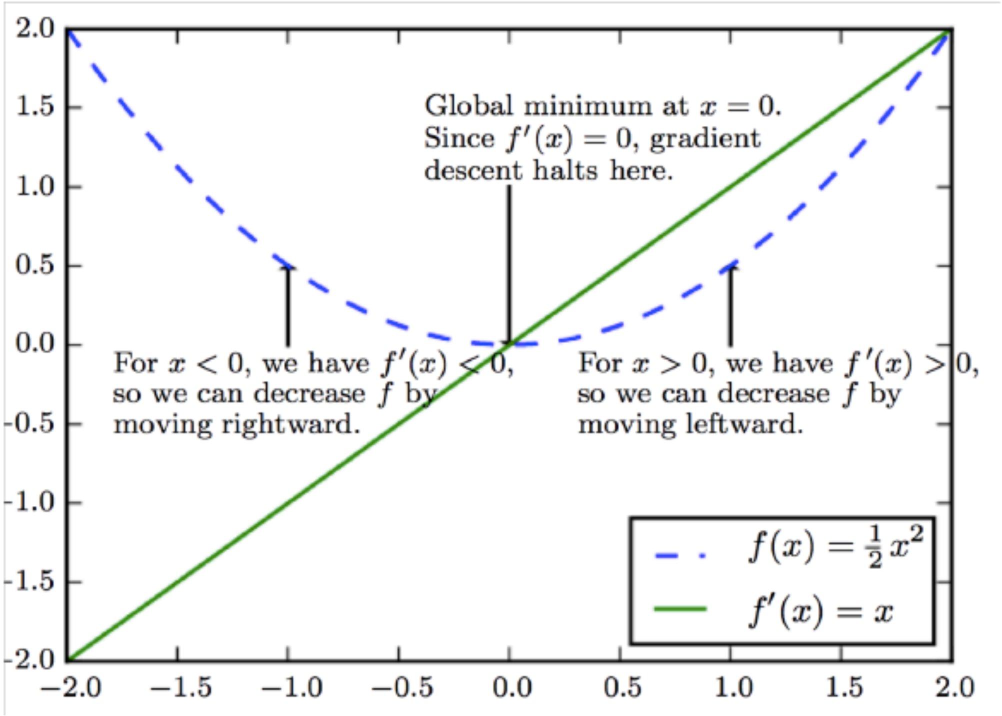


Figure 5 Gradient descent

Source: Srihari, S. (n.d.)

There is $f(x) = y$ which is the function, and $f'(x)$the derivative. x and y are real numbers. Minimising a function using the derivative is helpful as it tells how to modify x to make a slight improvement in y. When moving x in the opposite sign of the derivative, $f(x)$ can be reduced. This is also named the gradient descent. When $f(x) = 0$, this corresponds to critical points (also called stationary points). When $f(x)$ is lower than all adjacent points, it's called the local minimum. At this point, $f(x)$ cannot be decreased anymore. When $f(x)$ is higher than all adjacent points, it's called the local maximum. At this point, $f(x)$ cannot be increased by making small steps. Saddle points correspond to critical points that are neither maxima nor minima. The global minimum is a point that corresponds to the absolute lowest value of $f(x)$. When flat regions are surrounding saddle points or when many local minima are not optimal, optimisation of the function is performed in the practical use of deep learning. Steepest descent corresponds to the discrete analogue of gradient descent. Hill climbing is an approach like steepest descent that is used with large discrete problems (Russel and Norvig, 2003).

3.1.2 Recurrent neural networks

It was discussed earlier in the previous section that feedforward neural networks have connections that do not form cycles. Recurrent neural networks (RNNs) are obtained when cyclical connections are allowed. Elman networks (Elman, 1990), Jordan networks (Jordan, 1990), time-delay neural networks (Lang et al., 1990) and echo state networks (Jaeger, 2001) are varieties of RNNs. Recurrent neural networks (RNNs) processes sequential data. Information about what has been calculated so far can be memorised in RNN. The limitation is that they can look back only a few steps. Recurrent connections are a way to bind inputs to the current or previous system states. Figure 6 represents a simple RNN with a unique self-connected hidden layer.

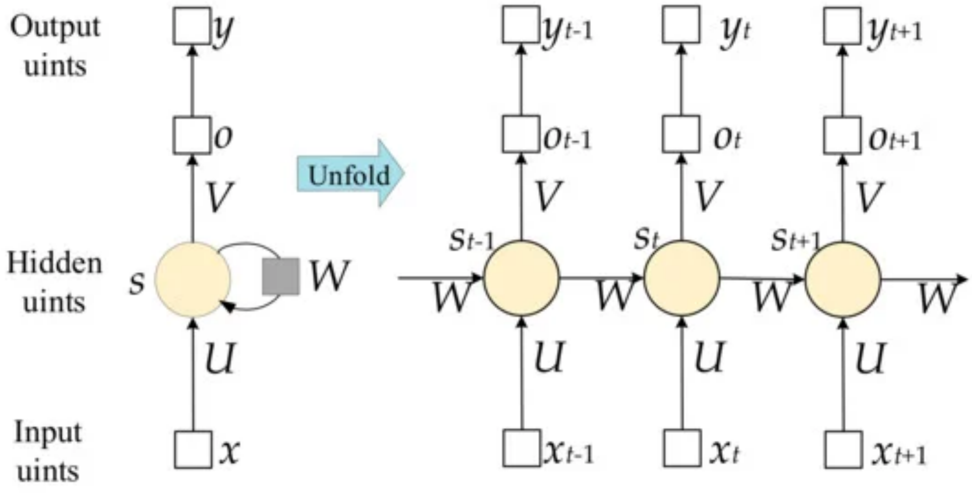


Figure 6 Recurrent neural networks structure.

Source: Li, Wang, Zhang, Xin, and Liu (2019)

More sophisticated types of RNNs have been developed over the years. Two RNNs stacked on top of each other create a bidirectional RNN. Based on the hidden state of both RNNs, the computation of the output can occur. Deep bidirectional RNNs are like bidirectional RNNs and give a higher learning capacity, but large training data is needed. LSTM networks are to RNNs regarding the architecture, but the hidden state is computed differently. Cells is another name for memory in LSTMs. The decision to keep or erase information is determined by these cells. The input, current memory and previous state are then combined by those cells. For more descriptions on recurrent neural networks, please refer to the textbook of Graves (Graves, 2012). RNN captures the sequential information present in the input data, i.e., the dependency between the words in the text while making predictions. RNNs share the parameters across different time steps. This is popularly known as Parameter Sharing. This results in fewer parameters to train and decreases the computational cost.

Deep RNNs (RNNs with many time steps) also suffer from the vanishing and exploding gradient problem, a common problem in all the different types of neural networks.

3.1.3 Convolution neural networks

Convolutional neural networks (CNNs) are also called convolutional networks. They are a neural network that works with grid-like input data and time-series data (1d grid). Time series means taking samples at regular time intervals. CNNs also work with image data (2-d grid). The variations in images cannot be considered with algorithms such as KNN. Convolutional neural networks architecture consists of three main layers: the convolutional layer, the pooling layer, and the fully connected layer. The convolutional layer is the core building block. For example, one image becomes a stack of filtered images in the convolutional layer using images. Pooling in the pooling layer is the process of shrinking the image stack. It is necessary to pick a window size, a stride, walk to window across the filtered images and from each window, the maximum value is taken. It is used for dimensionality reduction. This helps limit both the memory and processing requirement for running a CNN. The fully connected layer is the last layer, and it can classify data samples. Every value gets a vote. AlexNet is a solution that won the IMAGENET Challenge in 2012 using deep convolutional neural networks. IMAGENET is a challenge for evaluating algorithms for object detection or localisation and image classification from images and videos. AlexNet gave substantially better results than previous methods (Krizhevsky, Sutskever and Hinton, 2012). Figure 7 represents the AlexNet architecture. AlexNet consists of eight learning layers. It has five convolutional layers and three fully connected layers. The output of the final fully-connected layers is a softmax regression which converts the weights to probability distributions of the given 1000 classes. CNN learns the filters automatically without mentioning them explicitly. These filters help extract the right and relevant features from the input data. CNN also follows the concept of parameter sharing.

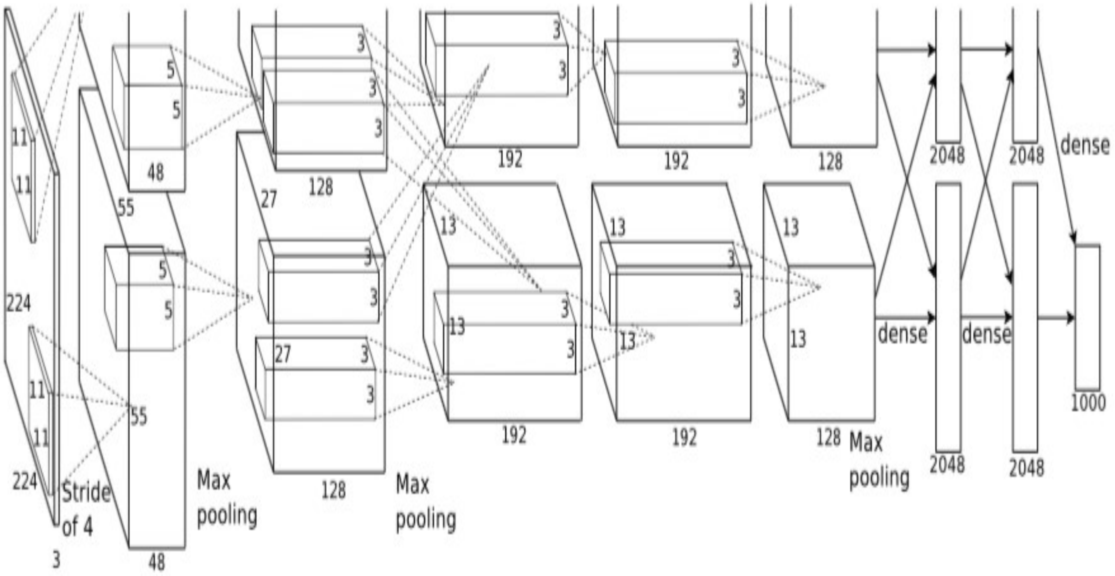


Figure 7 AlexNet CNN architecture

Source: Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2017)

3.1.4 Applications

To apply deep learning techniques successfully, a good knowledge of algorithms is not sufficient. A reputable machine learning practitioner must select an algorithm based on a specific application. There is also a need to monitor and respond appropriately to feedback observed during the experiments. The machine learning system will therefore be able to be improved. Gathering more data, increasing/decreasing the model capacity, improving inferences in a technique or debugging are important decisions that need to be performed. All those tasks are time-consuming, and it is essential to decide earlier what action to take. Recommendations can be adapted from Andrew Ng lecture at Stanford University (Ng, 2015). They determine the model's goals or what error metric/error metric target value to choose is part of standard recommendations. Those parameters should be selected depending on the problem that the application is trying to solve. Establishing an end-to-end working plan as early as possible in a project is also essential. Performance bottlenecks need to be determined, and the practitioner should diagnose which components are performing better or worse than expected. The practitioner should decide if it is due to overfitting, under-fitting, or a defect in the data. A final recommendation is the ability to make incremental changes. This can include gathering new data, adjusting parameters, or changing algorithms.

To give a few examples, deep learning can resolve problems in computer vision, speech recognition or natural language processing (NLP). Designing algorithms that can perform various tasks is one goal of deep learning, but specialisation is usually needed. Charges related to computer vision require modelling many input features (pixels). Tasks in NLP require modelling many possible values (as inputs features) corresponding to words in the vocabulary. Big data analytics is one of the recent applications of deep learning. Many private and public organisations have started to collect large amounts of information specific to their domains that can be useful to solve problems. It can be seen particularly in national intelligence, cyber security, marketing, fraud detection or medical informatics. Deep learning can analyse and learn large amounts of unsupervised data, useful in big data analytics. In Big data analytics, raw data is often unlabelled and un-categorized (Najafabadi, Villanustre, Khoshgoftaar, Seliya and Muharemagic, 2015). Najafabadi et al.'s paper cover the latest deep learning applications and the challenges in big data analytics. Nowadays, many companies provide deep learning solutions across various applications. The latest companies include Affectiva, Gridspace, Ditto Labs, Nervana, Deep Genomics, Indico, Deep Instinct, Clarifai, Ibidon, Enlitic, Metamind, Ripjar, MarianaIQ.

3.2 K-Nearest Neighbors

3.2.1 General concept

K-nearest neighbors algorithms are the most accessible algorithms to understand. Neighbors-based classification is classified as instance-based learning. Classification is obtained based on a majority vote of the nearest neighbors of each point. The data class of the point queried is obtained based on the data class with the most representatives within the nearest neighbors of that point. KNN is used for both classification and regression problems. An example of a KNN classifier is represented in figure 8 (Ajanki, 2007).

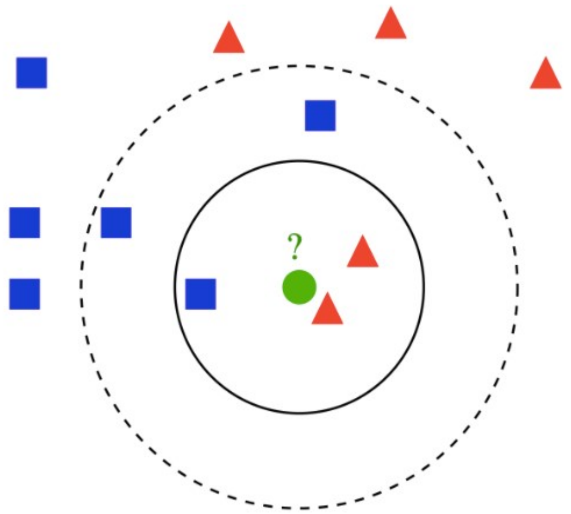


Figure 8 Example of k-nearest neighbors classifier

Source: Ajanki, A. (2007)

This is a KNN classifier with k=3 (solid line) and k=5 (dotted circle line). There are red and blue known samples and a green unknown sample. The samples are placed in a two-dimensional feature space. Each feature is one dimension to classify the unknown sample as red or blue; the classifier uses a distance function to find the k-nearest neighbors of the unknown sample. It can then predict the label of the green sample by finding the majority of red or blue labels among the k-nearest neighbors. In this example, when k=3, the unknown sample is predicted as red and when k=5, it is predicted to be blue.

3.2.2 Distance function

The distance function is an essential element of KNN algorithms. The performance of the KNN algorithm can depend significantly on the distance function. The most popular distance function is the Euclidian distance function. It is not used for categorical data. For categorical data, Hamming distance function is used. The euclidian distance can be generalised to a Minkowski distance (p-norm). For p=2, it corresponds to the Euclidian distance. For p=1, it corresponds to Manhattan distance. By changing values of p, it is possible to obtain an all family of distance functions with properties that are very different from the Euclidian distance.

3.2.3 Advantages and disadvantages

With nearest neighbors, there is no assumption. The only thing that is assumed is proximity. Similar instances should have similar class values or similar targets for regression. Assumptions are implied by the distance function. It's a non-parametric approach. In a sense, it lets the data speak for itself. There is nothing to infer from the data except k or possibly D. It is easy to update in an online setting by adding a new item to the training set. Cover and Hart (Cover & Hart, 1967) were able to show in the large sample case that the probability of error of a 1-nearest neighbors classifier is less than twice the Bayes error rate. Their paper also indicates that the error probability of the KNN rule monotonically decreases in k to the Bayes error probability showing the versatility of the family of KNN rules. Another advantage is that KNN is particularly well suited for multi-modal classes and applications in which an object can have many class labels. On the downside, it is necessary to handle missing data by filling in or creating a unique distance. It is sensitive to class outliers (mislabelled training instances). It is also sensitive to many irrelevant attributes (affect distance adds noise to the distance function). It is like Naïve Bayes in this case and not like decision trees. Decision trees will ignore irrelevant attributes. This algorithm is not learning anything, and it is just storing all the training instances and then comparing them at the testing time. It means that there is a need for space/storage to keep all training examples and time to compute the distance to all examples. If the number of training examples increases, the system will become slower and slower. Expense is testing, not training time. It is a problem as it is better to have large training sets to make reasonable and accurate estimates.

3.3 Support Vector Machines

3.3.1 History of SVMs

Support Vector Machine is a supervised learning technique used for classification and regression problems. It was first introduced in a 1992 article (Boser et al., 1992) in which the author presented a training algorithm. The margin was maximised between the decision boundary and the training patterns. SVMs were put into practical application as Large Margin Classifiers. Vladimir Vapnik et al. (Vapnik, 1999) discussed the Statistical Learning Theory about SVM. The statistical learning theory discusses the problem of choosing desired functions based on empirical data. Structural Risk Minimization (SRM) is the basis of SVMs. It was introduced by Vapnik et al. SVMs became successful within handwritten digit recognition (Bottou et al., 199The article compares the performance of multiple classifiers algorithms using handwritten digits from a standard database 1.1% test error rate was discovered using SVM. This is the same error rate of a carefully constructed neural network that they refer to as LeNet4. With SVMs, the number of training samples is not impacting the results (Jonsson et al., 2002). When most neural networks compare the empirical risk minimisation principle, better results are obtained. (Juwei Lu et al., 2001). For the estimation of how various features modify classification results and identifying the one that is important in planning, SVMs can be helpful. This is in addition to classification. SVMs have been successfully utilised in face detection, image detection, speech recognition or prediction. Bankruptcy prediction is an example. There are other uses (Byun et al., 2003).

3.3.2 SVM concept

The goal of SVMs is to minimise an upper bound of the generalisation error by maximising the margin between the data and the separating hyperplane (Amaris and Wu, 1999). Neural networks minimise the empirical training error. Empirical Risk Minimization (ERM) corresponds to standard learning approaches to reduce error on the training dataset. A neural network is a typical example of an ERM. As described earlier, the basis for SVM is

Structural Risk Minimization (SRM). SRM minimises an upper bound on the expected risk. ERM minimises the error on the training data. SVMs can generalise well because of this difference, and it's the goal to achieve in statistical learning. There is no overgeneralisation with SVM (Mitchell, 1997).

3.3.2.1 Support Vector Machines: The linearly separable case

The hyperplane which corresponds to a decision plane is what's behind SVMs. The positive class (+1) and the negative class (-1) are separated with the largest margin. The latest is associated with the Vapnik-Chervonenkis (VC) dimension of an SVM. The VC dimension quantifies the capacity (expressive power, complexity, flexibility, or richness) of a space of functions that a statistical classification algorithm can learn. It corresponds to the cardinality of the largest set of points that the algorithm has shattered. When two classes are linearly separable, finding a hyperplane that gives the smallest generalisation error compared to all the possible hyperplanes is not necessary. This hyperplane is the hyperplane with the maximum margin of separation between the two classes. The sum of the distances from the hyperplane to the closest data points of the two classes corresponds to the margin. Support Vectors (SVs) refer to as the closest points. The solid line represents the optimal separating hyperplane in figure 9. The linear separable case is also referred to as the linear hard-margin classifier.

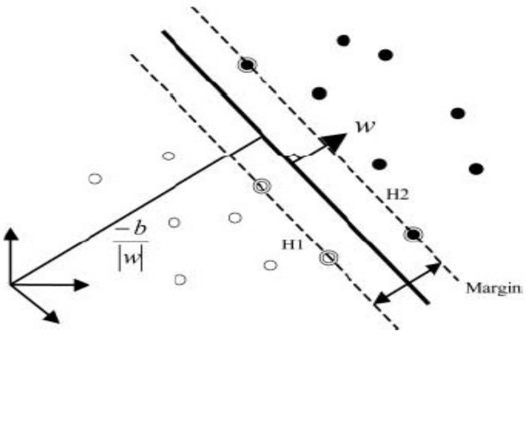


Figure 9 Linear separating hyperplane: Separable case

Source: Burges, C.J.C. (1998)

3.3.2.2 Support Vector Machines: The linearly non-separable case

With data used in practical applications, the separation between the two classes is incomplete, but a hyperplane can still be defined. The hyperplane maximises the margin of the training data. Positive slack variables in the constraints make this possible. In an optimisation formed to equality, when a slack variable is an added problem, inequality constraint is trans. The non-separable case is also referred to as the linear soft-margin classifier and is represented in figure 10.

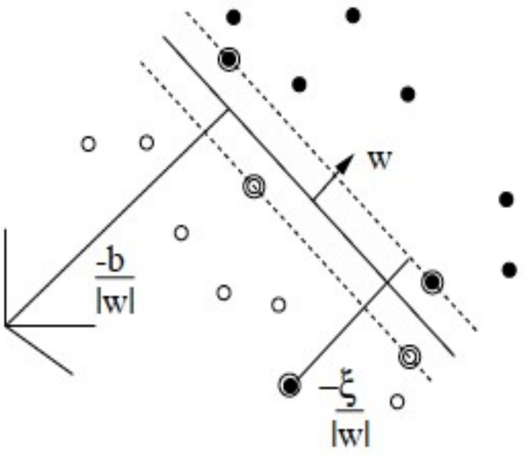


Figure 10 Linear separating hyperplane: Non-separable case

Source: Kumar, S. and Apparao, M. (2017)

3.3.2.3 Nonlinear Support Vector Machines

Nonlinear decision surfaces modifications are mandatory in classification problems when a linear classifier is used (Tay and Cao,2002). If the linear decision surface does not exist, the data is mapped to a higher-dimensional space. It is also called the feature space, where the separating decision surface can be found. In figure 11, the generalised optimal separating hyperplane is represented and constructed in the high dimensional feature space.

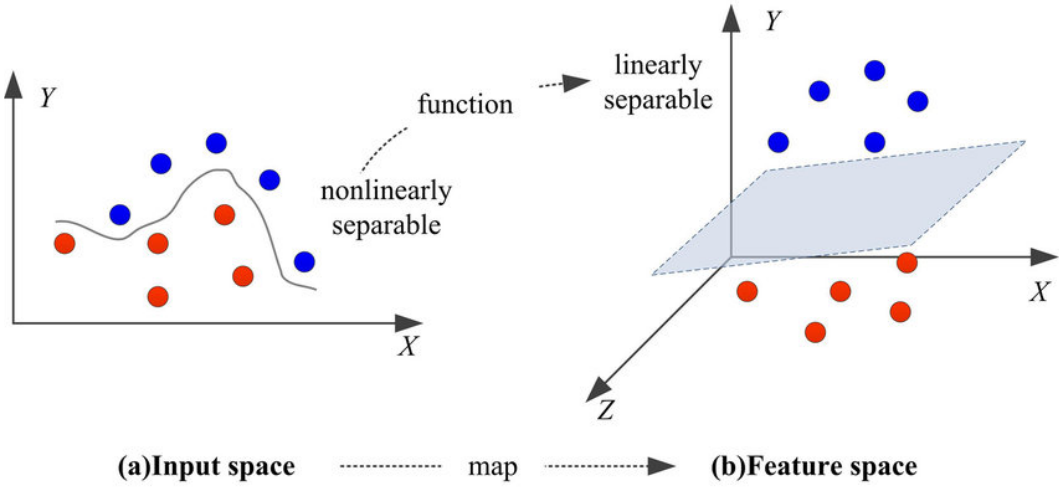


Figure 11 Input space and feature space

Source: Cheng, C.-T., Feng, Z.-K., Niu, W.-J. and Liao, S.-L. (2015)

Computational learning theory makes use of Cover’s theorem. When nonlinear transformation and a high enough dimensionality of the feature space is present, there is a transformation of the input space into a new feature space. The patterns are linearly separable with high probability in this feature space (Haykin, 2009). The nonlinear transformation is obtained with the so-called kernel functions.

Kernel substitution is a way to obtain nonlinear algorithms from algorithms that had previously a restriction on handling linear separable datasets (Campbell, 2000). Using implicit kernels permit bypassing the curse of dimensionality (Vapnik,1999). Various learning machines are built based on different kernel functions and create different hyperplanes in the feature space.

3.3.2.4 Quadratic programming problem of SVMs

Quadratic programming is a type of mathematical optimisation problem. Quadratic programming is the problem of finding a vector x that minimises a quadratic function, possibly subject to linear constraints.

In the previous paragraph that discussed the linearly separable case, it was defined that the optimal separating hyperplane is achieved with the minimisation of an equation under a specific constraint to separate the training data correctly. The optimisation goal used in the equation is quadratic. The constraints are linear. Based on this constrained optimisation problem, there is a possibility to create another problem. It's called the dual problem. Duality is sometimes called the duality principle. It refers to optimisation problems that can be observed from two different perspectives that correspond to the first problem and the dual problem. The dual problem solution leads to a lower bound to the primal problem or minimisation solution. The dual problem is also named the Lagrangian dual problem. Oher dual problems can be used, such as the Fenchel dual problem or the Wolfe dual problem. The dual problem can be formulated using the Lagrange multipliers: With the available training sample, find the Lagrange multipliers that maximise the objective function subject to specific constraints. The objective function that needs to be maximised in the situation of non-separable problems in the dual problem is almost equivalent to the case for the separable problems. The difference is that the constraints in the separable case are switched with more stringent constraints in the non-separable case.

There are several suggestions regarding the algorithms that can solve the dual problems. QP algorithms that were used traditionally (Schölkopf, Burges and Smola, 1999), (Smola and Schölkopf, 2004) are not the most appropriate when problems are significant for few reasons (Keerthi, Shevade, Bhattacharyya and Murthy2001) because the kernel matrix must be computed and stored in memory. The consequence is that a large memory is needed. Methods used include the Cholesky decomposition of a large submatrix of the kernel matrix.

Finally, coding these algorithms is very difficult, especially for practitioners developing their SVM classifier implementation. Some methods can get rid of some or sometimes all the problems defined earlier. A whole new set of QP problems was suggested by Osuna et al. (Osuna, Freund and Girosit, 1997). A series of smaller QP sub-problems creates the significant QP problem. One example that violates the Karush-Kuhn-Tucker (KKT) conditions must be added to the examples for the previous sub-problem. In mathematical optimisation, first-order necessary conditions (KKT conditions) are conditions so that a solution in nonlinear programming is optimal, including the satisfaction of some regularity conditions. A sequence of QP sub-problems will be guaranteed to converge when there is the addition of a least one violator all the time (Osuna, Freund and Girosit, 1997An optimisation new training algorithm named Sequential Minimal Optimization was proposed by Platt (Platt 1998). It solves the SVM QP problem rapidly without the need for extra matrix storage and the use of numerical QP optimisation steps. SMO divides the QP problem into multiple QP sub-problems using Osuna's theorem. Two Lagrange multipliers are chosen by the SMO to optimise at each step jointly. After finding the optimal values for these multipliers, SVMs are updated to indicate the new optimal values. Numerical QP optimisation is avoided as the two Lagrange multipliers are solved analytically. Extra matrix storage is not needed; therefore, the memory of a personal computer is sufficient for significant SVM training problems (Platt, 1998). Using a single threshold in Platt's SMO algorithm can lead to confusion and inefficiency, as Keerthi et al. (Keerthi, Shevade, Bhattacharyya and Murthy, 2001) pointed out. Using clues taken from the KKT conditions and applied to the dual problem, two threshold parameters are used to generate changes of SMO. The basic SVMs for two-class problems initially separated the binary classes (k=2) with a maximised margin criterion (Cortes and Vapnik, 1995).

However, real-world problems often require discrimination for more than two categories. Multi-class pattern recognition has a wide range of applications, including optical character recognition (Mori and Suen, 1995), intrusion detection (Khan, Awad and Thuraisingham, 2007), speech recognition (Ganapathiraju, Hamaker, Picone, 2004) and bioinformatics (Baldi and Pollastri, 2002). In practice, the multi-class classification problems (k>2) are commonly decomposed into a series of binary problems such that the standard SVM can be directly applied.

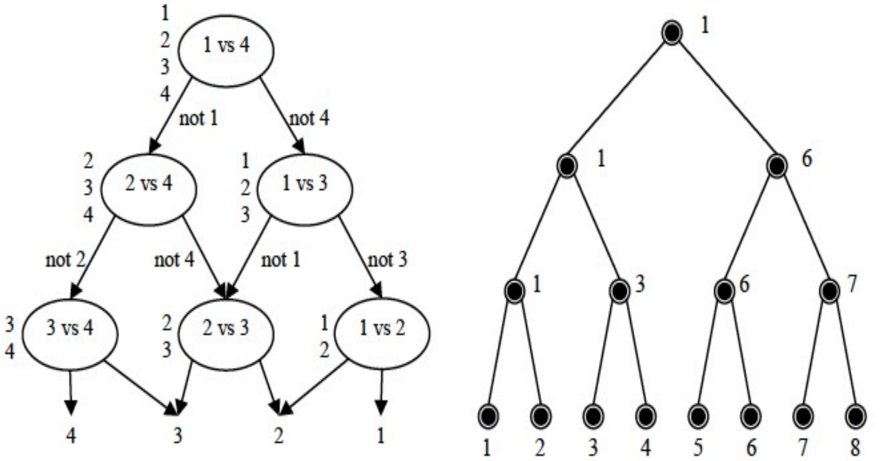
Two representative ensemble schemes are one-versus-rest (Vapnik, 1998), (Bin, Yong and ShaoWei, 2000) and one-versus-one (Kreßel,1999). Both one-versus-rest and one-versus-one are special cases of the Error-Correcting Output Codes (ECOC) (Dietterich and Bakiri, 1995), which decomposes the multi-class problem into a predefined set of binary problems. Multi-class SVMs are discussed in the following sub-chapters.

3.3.2.5 Multi-class SVMs: One to others

One to others SVM is the method1 described in the article published by Bin et al. (Bin, Yong, and Shao-Wei, 2000). The first labelled samples can be classified by SVM1 and the ith labelled ones by SVM. For the in-class problem (n>2), SVM classifiers are referred as SVMi, i=1,2,…n.

3.3.2.6 Multi-class SVMs: Pairwise SVMs

Pairwise SVM is the method2 described in the article published by Bin et al. (Bin, Yong, and Shao-Wei, 2000) and is also mentioned in the article published by Kreßel (Kreßel, 1999). Trees correspond to the arrangement for pairwise classifiers. An SVM is represented by each tree node. Pontil et al. (Pontil and Verri 1998) suggested a bottom-up tree to recognise 3D objects. It was also applied to face recognition (Guo, Li and Chan, 2000), (Guo, Li and Chan, 2001). A recent publication discusses the top-down tree structure (Platt, Cristianini and Shawe-Taylor, 1999). The performance of the two strategies with classification problems is not analysed theoretically (Heisele, Ho, Wu and Poggio, 2003). New experimentation on people recognition indicates identical performances for classification for both strategies (Nakajima, Pontil and Poggio, 2000). Figure 12 represents examples of tree structures of multiclass SVMs.



(a) example of top-down tree structure (b) example of bottom-up tree structure

Figure 12 Multi-class SVMs: Tree structure

Source: Byun, H., Lee, S. W. (2003)

3.3.3 Applications of Support Vector Machines

It was discussed earlier that beta-barrel membrane topology prediction is considered a pattern recognition problem. In this section, a discussion on some of the use of SVMs for pattern recognition problems is addressed. Face detection, verification and authentication are very popular in biometric, access control, identity authentication, or surveillance in general. Much current research is performed for those applications using different methods. Reliable performance is challenging to achieve. Identical facial configurations can be observed with some persons due to minimal variations in the light intensity, makeup on faces, different poses, and facial expression. Using a pair of glasses or having moustaches can make it harder to recognise faces (Wang, Chua and Ho, 2002). Osuna et al introduced frontal face detection (Osuna, Freund and Girosit, 1997). A 19x19 window scan input images and an SVM is trained with a new decomposition algorithm and 2nd-degree polynomial kernel function. Global optimality can be guaranteed. Kumar et al. (Kumar and Poggio 2000) have used the Osuna et al. algorithm to track real-time faces. The algorithm's speed was improved by Romdhani et al. (Romdhani, Schokopf, and Blake, 2001) using a method that includes reduced support vectors. They are derived from support vectors. The improvement of face detection performance was obtained with majority voting as described in the paper written by Bassiou et al. (Bassiou, Kotropoulos, Kosmidis and Pitas, 2001). For face recognition and authentication, it comes to two different problems to solve. In face recognition problems, a face is used as a test, and multiple references faces are within a database. The algorithm tries to find the most significant number of similar reference faces to the test face. A face for test and a face for reference are used in face authentication problems. The algorithm needs to decide if the face for the test is the same as the face used for reference.

Object detection and recognition have the objective of finding and tracking people on the move. Applications include surveillance or traffic control. A system using SVM was suggested by Pittore et al. It can detect the presence of people on the move from an image sequence (Pittore, Basso and Verri, 1999).

Handwritten character/digit recognition is also another application of SVMs. SVMs have better performances than any other learning algorithms. There are problems with handwriting recognition as considerable variability and distortions of patterns. Two different feature families (statistical and structural) with an SVM classifier have been used by Gorgevik et al. for handwritten digit recognition (Gorgevik, Cakmakov and Radevski, 2001). Single SVM classifiers have better performance than rule-based reasoning applied to two separate classifiers. A vision-based handwritten digit recognition system using an SVM classifier has been created by Teow et al. (Teow and Loe, 2002). SVM classifiers have been utilised in Chinese check recognition systems, and Bin et al. (Bin, Yong, and Shao-Wei, 2000) demonstrated that SVM possesses better generalisation ability than other classifiers.

Speaker and speech recognition is another application of SVMs, and they have been used with different datasets. The standard thresholding rule was replaced with SVMs for speaker verification decision, acceptance, or rejection (Bengio and Mariethoz, 2001). There is a significant improvement in performance with text-independent tasks. The normalisation of polynomial kernel within SVMs was proposed by Wan et al. (Wan and Campbell, 2000).

Content-based image retrieval is another application of SVM. It can be used for a digital library or a multimedia database. A distance-from-boundary for retrieving the texture image was suggested by Guo et al. (Guo, Li and Chan, 2001). SVM can provide the boundaries between classes.

In bioinformatics, significant applications include identifying proteins functions, gene functions and microarray classification. Identifying protein function includes secondary structure prediction, identification of binding sites, sub-nuclear localisation of proteins, subcellular localisation, protein-protein interaction prediction, prediction of protein disorder, identification of gene function includes promoter prediction, prediction of tissue-specific localisation of genes, prediction of DNA methylation sites and DNA hot spots prediction. Microarray classifications include leukaemia prediction, colon cancer prediction and prediction of several genetic disorders. Data fusion which corresponds to heterogeneous biological data is bringing more attention. The human genome is almost entirely sequenced. Given a specific gene, there is a possibility to know the protein it encodes, to understand how similar that protein is compared to other proteins, how mRNA expression levels are associated, the frequency of known or inferred transcription factor binding sites that can be found in this gene's upstream region or to know the identities of multiple proteins that have an interference with the gene's protein product. Pavlidis et al. (Pavlidis, Weston, Cai and Noble, 2002) demonstrated the application of the SVM learning algorithm to understand cellular function at the molecular level by grouping information from disparate types of genomic data. SVMs were trained to recognise the function category of yeast genes using an ensemble of microarray gene expression data and phylogenetic profiles. Data are fixed-length, real-valued vectors in both types and a third-degree polynomial kernel have been used. There is a comparison between the three different techniques for combining those two data types. The comparison is represented in figure 13.

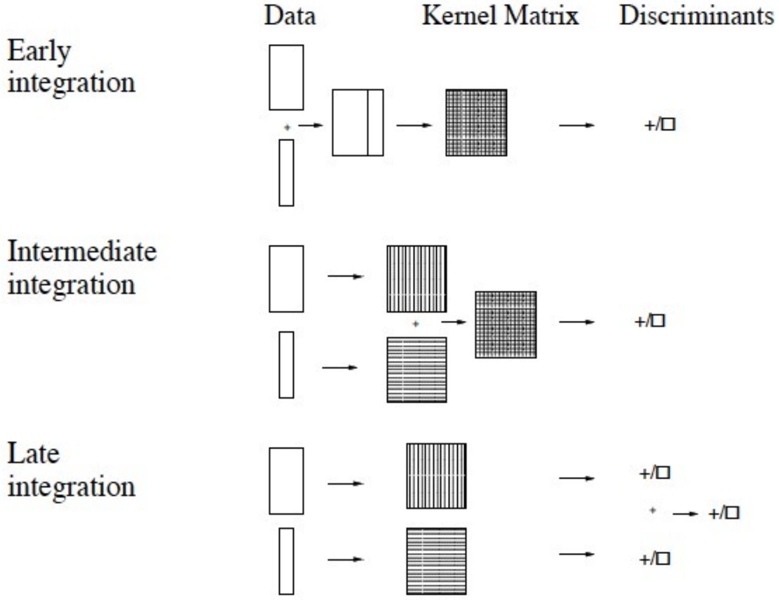


Figure 13 Three methods for learning from heterogeneous data using SVM

Source: William Stafford Noble (2017)

Two data types are linked together to generate a specific set of input vectors for early integration. The kernel values are being calculated separately for each data set and then calculated with a sum in intermediate integration. The SVM trains each data type. The discriminant values are obtained with a sum in late integration. This paper also presents heuristic techniques regarding scaling factors in the application for each kernel function.

3.3.4 Limitations of Support Vector Machines

Kernels have a significant impact on SVM performances. An explanation of the relation between the standard regularisation theory and the SVM kernel method was given by Smola et al. (Smola, 1998). Amari et al. (Amari and Wu, 1999) suggested a kernel having a modification based on information-geometric consideration of the structure of the Riemannian geometry induced. The suggestion had the objective to increase performance. Sizes of the training and testing phases are also significant. When massive datasets that can include millions of support vectors are used in the training phase, there is a problem that has not been solved (Burges, 1998). A modification of the SMO (Platt, 1998) was proposed to solve the training problem.

Another complex problem can be related to controlling the selection of Support Vectors. If patterns that need to be classified are non-separable and the training data are noisy, there is often a problem. Removing known errors from the data before training or removing them after training will not produce the same optimal hyperplane. It's because the errors are necessary for penalising non-separability (Haykin, 1998). Limitations of SVM will also be discussed while interpreting results from the MATLAB implementation.

Some elements used to implement an ANN and KNN can be reused to implement an SVM classifier. The datasets, binarisation of the inputs/outputs, construction of the input and target matrices, data division and assessPerformance function used previously will be similar.

3.4 Ensemble methods

Ensemble's methods use various machine learning algorithms to obtain better predictive performance than the learning algorithms alone. It is not the same as statistical ensemble used in statistical mechanics. Statistical ensembles are most of the time infinite. Machine learning ensembles consist of a concrete finite set of alternative models. It has been decided to use an ensemble method as part of this paper to improve the predictive performance. Cascading is a unique model of ensemble learning that is based on the concatenation of several classifiers. It uses all information collected from the output from a given classifier and use it as input for the next classifier in the cascade. Compared to voting or stacking ensembles with multi-expert systems, cascading is multistage.

3.4.1 General concept

An ensemble is a supervised learning algorithm. It is trained and then will be used to make predictions. The trained ensemble represents a single hypothesis. Ensembles can be represented by having more flexibility in the functions they can represent. This flexibility can lead to overfitting of the training data. The ensemble techniques such as bagging tend to reduce problems that correlate with the overfitting of the training data. Ensembles can provide excellent results when there is diversity among the models (Kuncheva and Whitaker, 2003) (Sollich and Krogh, 1996). Most ensemble techniques look to include variety among the models they are combining. (Brown, Wyatt, Harris, and Yao, 2005).

The number of component classifiers for an ensemble impacts the accuracy of prediction. Few studies have been addressing this issue. For online ensemble classifiers, it’s even more critical to determine the ensemble size, volume, and velocity of significant data streams. Statistical tests have been used to get the correct number of components. A new theoretical framework suggests that an ideal number of component classifiers for an ensemble can be found. The framework is named “The law of diminishing returns in ensemble construction”. This framework indicates that the highest accuracy can be achieved using the same number of independent component classifiers as class labels (Bonab and Can, 2016).

Classification ensembles can be found in the Statistics and Machine learning Toolbox (Mathworks, 2017). It includes boosting, random forest, bagging, random subspace and ECOC ensembles for multiclass learning.

3.4.2 Types of ensembles

3.4.2.1 Voting and averaging based ensemble methods

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms. Voting has been used mostly for classification, and averaging has been used for regression. Creating multiple classification/regression models using training datasets is the first step for both approaches. Each base model will be created using various splits of the same training dataset and same algorithm. It can also be created using the same dataset with different algorithms. Every model makes a prediction, all called a vote, for each test instance. The final output prediction is the one that receives more than 50% of the votes in majority voting. If none of the predictions gets more than 50% of the votes, it can be concluded that the ensemble method cannot make a stable prediction for this instance. There is a possibility to try the most voted prediction as to the final prediction, and it's named 'plurality voting' in some papers. Weighted voting is another option that can be used. In majority voting, each model will have the same rights. In weighted voting, it's possible to emphasise one or more models. The prediction of the better models is counted multiple times in weighted voting. Finding an appropriate set of weights is up to the person performing the task. The average predictions will be calculated for every instance of the test dataset in the simple averaging method. Most of the time, this method will decrease overfitting and will create a smoother regression model. The weighted average is the last method available and corresponds to a modified version of simple averaging. The average is calculated from the multiplication of the prediction of each model by the weight.

3.4.2.2 Bootstrap aggregating (bagging)

Bootstrap aggregating is also named bagging, is composed of each model in the ensemble vote with equal weight. In bagging, training is done on each model in the ensemble using a subset of the training dataset selected randomly. The first step involves the creation of multiple models. The models will be generated using an identical algorithm with random sub-samples of the dataset. These sub-samples are selected randomly from the original dataset using the bootstrap sampling method. Each sub-sample will be created independently from the other. The training and generation can be executed at the same time. The bagging technique is used in the random forest algorithm.

3.4.2.3 Boosting

Boosting defines a group of algorithms that can convert weak models to strong models. Boosting is building an ensemble and will train each model with the same dataset. The weights of instances will be modified and adjusted following the error of the last prediction. The models are then obliged to focus on the complex instances. Boosting is a sequential method, and parallel operations will not be used compared to bagging where it's possible. Adaboost is a popular boosting method algorithm. The Gödel prize was won by the creators of Adaboost. It can be executed simultaneously with multiples other learning algorithms to improve performance. The output of the different learning algorithms, also named weak learners, will be grouped into a weighted sum. That sum represents the final output of the boosted classifier. AdaBoost can be adapted, meaning that subsequent weak learners are slightly modified in favour of those instances that were not classified correctly by previous classifiers. AdaBoost is also sensitive to noisy data and outliers. In some situations, it will be less sensitive to the overfitting problem than other learning algorithms. Individual learners can be weak; however, if their performance is slightly better than random guessing, the final model will converge to a strong learner. Each learning algorithm will tend to fit specific problem types more than others and have various parameters and configurations to be modified before obtaining optimal performance on a dataset. AdaBoost, including decision trees as weak learners, is often considered the best out-of-the-box classifier (Kégl, 2013). When it is used in conjunction with decision tree learning. The information obtained at each stage about the relative 'hardness' of each training sample is put into the tree algorithm. Later trees will tend to focus on harder-to-classify samples.

3.4.2.4 Multi-stage method cascading

Cascading is a multistage method. Associated with each learner is confidence $w5$ such that it can be said that $d5$ is confident of its output and can be used if $w5$ > θj (the threshold). Confidence corresponds to misclassifications and the instances for which the posterior is not high enough. The idea is that an early simple classifier handles most instances, and a more complex classifier is used only for a small percentage, so it does not significantly increase the overall complexity. A Multi-stage Deep Classifier Cascades (MDCC) is represented in Figure 14 to address some challenges in the open world recognition scenario. Open world learning (OWL) is also known as open-world recognition, classification, or open-world AI. It is getting increasingly important as the learning agent is increasingly working in or facing the real-world open and dynamic environment. The core of open-world learning or open-world AI is about recognising unknowns and learning them so that the AI agent will become more and more knowledgeable.

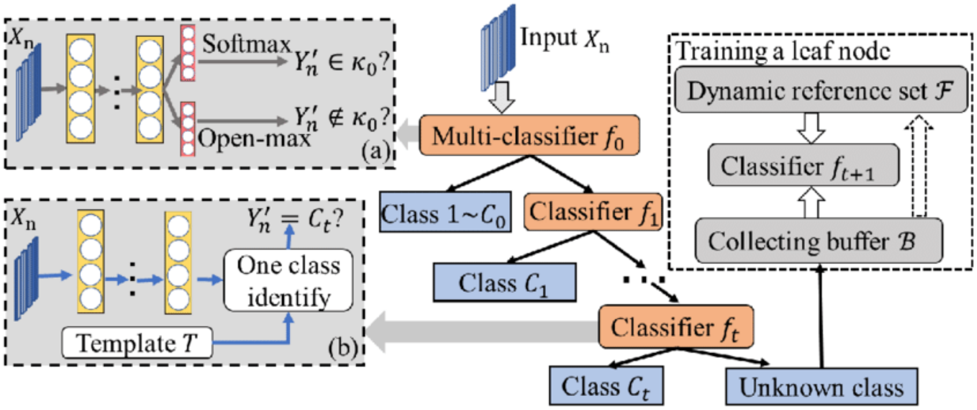


Figure 14 Architecture of Multi-stage Deep Classifier Cascades and model details of (a) root node and (b) leaf nodes

Source: Guo, X., Alipour-Fanid, A., Wu, L., Purohit, H., Chen, X., Zeng, K. and Zhao, L. (2019)

The first cascading classifier was a face detector. This classifier needed to be quick as it was implemented on cameras and phones with low-power CPUs (Viola and Jones, 2001). The term cascading classifier has been used in statistics to define a model with various stages. For example, a classifier such as k-means will use a vector of features and generate outputs for each possible classification result the probability that the vector belongs to the class. This can be mainly used to decide; however, cascading classifiers utilise the output as the input to another model, which corresponds to another stage. This will be useful for models that have highly combinatorial or counting, which cannot be fitted without looking at the interaction terms. The successive stage can gradually make an approximation of the combinatorial nature of the classification or add interaction terms in classification algorithms.

3.4.2.5 Stacking

Stacking is an ensemble learning technique to combine multiple classification models via a meta-classifier. The individual classification models are trained based on the complete training set; then, the meta-classifier is fitted based on the outputs -- meta-features -- of the individual classification models in the ensemble. The meta-classifier can either be trained on the predicted class labels or probabilities from the ensemble

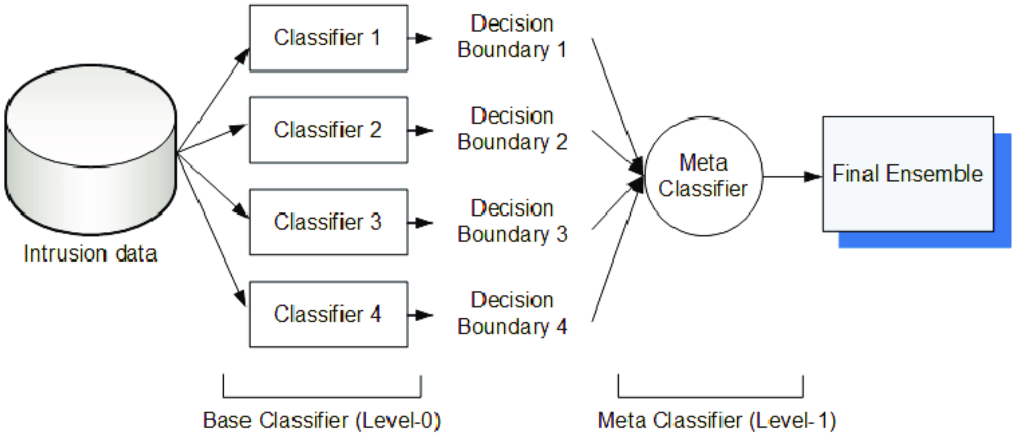


Figure 15 Stacking

Source: Tama, A.B., Rhee, K.H., (2017)

Tama et L. conducted a comparative experiment using different ensemble approaches, including stacking, in a recent paper. To prove classifier ensemble can perform on intrusion detection, they considered two real public datasets, e.g., network-based intrusion detection, namely NSL-KDD dataset (Tavallaee, M., Bagheri, E., Lu, W. and Ghorbani, A.A, 2009). and 802.11 network-based intrusion detection, namely GPRS dataset (Vilela, D.W.F.L., Ferreira, E.W.T., Shinoda, A.A., de Souza Araujo, N.V., de Oliveira, R. and Nascimento, V.E.,2014). Among classifier ensembles, stacking is a powerful method for IDS since it yields the best performance in terms of accuracy, precision, and F1.

3.4.3 Applications of ensembles methods

There have been various applications of ensemble learning in bioinformatics. Some examples include the classification of gene expression microarray data and MS-based proteomics data. Gene-gene interaction and identification using single nucleotide polymorphism (SNPs) data taken from Genome-Wide Association studies is another example. Prediction of regulatory elements from DNA and protein sequences is a third example. Ensemble methods can be applied to other bioinformatics problems apart from the three main areas listed earlier. For gene function prediction, a meta-ensemble based on SVM was suggested (Guan et al., 2008). It has three base classifiers. The prediction is achieved by selecting the best performance of the three for each gene ontology term. Shen and Chou (Shen and Chou 2006) created nine sets of features for an ensemble used to recognise protein folding. The features taken from the protein sequences included secondary structure, polarizability, hydrophobicity, polarity, Van de Waals volume, and various dimensions of pseudo-amino acid composition. KNN base classifiers have been trained with the use of different feature sets. They have been put together as part of a weighted voting manner. Ouali et al. (Ouali and King 2000) designed a classifier used for protein secondary structure prediction. It was created using cascading various types of classifiers using neural networks and linear discrimination. Melvin et al. (Melvin et al., 2008) introduced a combination of KNN classifier with an SVM classifier for protein structure prediction using sequence information. The KNN classifier was trained using global sequence information, also named full coverage. The SVM was trained with the use of local sequence information. Wang et al. (Wang et al., 2006) made the use of stacked generalisation for the prediction of membrane protein types. An SVM and a KNN have been used and correspond to the base classifiers. A decision tree was used for the combination of the base classifiers. The Ensemble technique was looked at for the protein-protein interaction problem. Lately, Deng et al. (Deng et al., 2009) made the use of an ensemble algorithm using weighted voting strategy and bootstrap resampling. The tricky part of this learning task corresponds to the imbalance of the data classes because of the lack of positive training examples. The authors found that the ensemble could reduce the imbalanced problem and significantly increase prediction performance. Ensemble methods have been used in various new studies that explain genetic networks. Wu et al. (Wu et al., 2010) suggested using a relevance vector machine (RVM) based ensemble in the prediction of human functional genetic networks out of multiple sources of data. The ensemble appears to be effective even with large missing values. The applications of ensemble methods in bioinformatics that have been described so far are not complete, but the main topics have been discussed. Ensemble methods are beneficial in general. It's flexible, and there are multiple ways to create and adjust them. The use of ensemble techniques for old and new biological problems will likely be the focus in the coming years. There is much promise around the ensemble method. Various extensions have been suggested. The following section focuses on multiple extensions that can be used to achieve better prediction.

An easy method to use SVM as part of the ensemble framework is applying the bagging procedure with the base classifier SVM. Caragea et al. (Caragea et al., 2007) took this approach. A bagging ensemble with the base classifier of SVM was used for glycosylation site prediction. The results showed that the SVM ensemble's performance suppressed both the single SVM and the balanced when they trained each base classifier with a re-sampling of the balanced training set. Guan et al. (Guan et al., 2008) used a bagging procedure to create an ensemble of SVMs in a gene function prediction problem. The ensemble of SVMs always performs better than the single SVM classifier around gene ontology term recognition., and robotics. Peng (Peng, 2006) looked at the concept of over-generating and selecting an appropriate subset of base classifiers. The bootstrap sampling method and the utilisation of a base classifier such as the SVM are used to create multiple training sets. There is more stability with the SVM for small perturbation of the training samples than the decision tree. If base classifiers need to be diversified, using a clustering-based base classifier selection procedure can ensure that the accuracy of base classifiers is strong even though they disagree with each other. When comparing a single SVM classifier to the ensemble of bagging and boosting, Peng concluded that the best results could be obtained with a clustering-based SVM ensemble. Gordon et al. (Gordon et al., 2006) suggested an approach to an ensemble where SVM has a different kernel and are combined for transcription start sites prediction. The approach defined by Gordon et al. gave a new way to create an SVMs ensemble. This could be very helpful for problems with heterogeneous data sources and feature types. It was looked at if an improvement was possible when creating a meta ensemble, an ensemble of ensembles. Dettling (Dettling 2004) looked at it and proposed a combination of bagging and boosting algorithms for microarray data classification. It is also called BagBoosting. It was thought that enabling ensemble has a low bias; however, the variance is high, and the bagging ensemble has lower variance and no changed bias. BagBoosting demonstrated that it could improve prediction compared to bagging and boosting alone. It is also very efficient compared to various SVM or k- NN classifiers. Guan et al. (Guan et al., 2008) suggested three different SVMs ensembles as base classifiers and are further combined as a meta-ensemble of SVMs in the use of gene function prediction. The final prediction of genes was obtained from selecting the best performing classifier based on each gene ontology term. Liu and Xu (Liu and Xu, 2009) looked at another option of creating meta-ensembles. The genetic programming approach was the basis for their ensemble system. Their experimentation showed that their system performed better than various other evolutionary-based algorithms. Some other methods attempted to look for the diversity of the base classifier with the use of heterogeneous classification algorithms. Bhanot et al. (Bhanot et al.,2006) used a combination of ANN, SVM, Weighted Voting, k NN, decision trees and logistic regression in mass spectrometry data classification. Kedarisetti et al. (Kedarisetti et al., 2006) used an ensemble for protein structural class prediction. A set of fifteen classifiers was combined in the paper from

Hassan et al. (Hassan et al., 2009). They were rule-based classifiers including k-NN and decision trees and function-based classifiers including SVM and neural networks. Their ensemble was applied to three microarray datasets to find a small number of highly differentially expressed genes. For microarray analysis, Yang et al. (Yang et al., 2010a) suggested a multi-filter enhanced genetic ensemble system. The system combines various classifiers and filtering algorithms with a genetic algorithm. Yang et al. (Yang et al., 2010b) used the genetic ensemble system for gene-gene interaction identification from Genome-Wide Association studies. Data-level perturbation can be combined with different classification algorithms to produce a meta-ensemble of classifiers. It could increase the overall diversity and provide higher classification accuracy. Figure 15 illustrates this type of ensemble method. For (a), classification algorithms have been trained using the same training set. For (b), classification algorithms have been prepared using different perturbations on the training set.

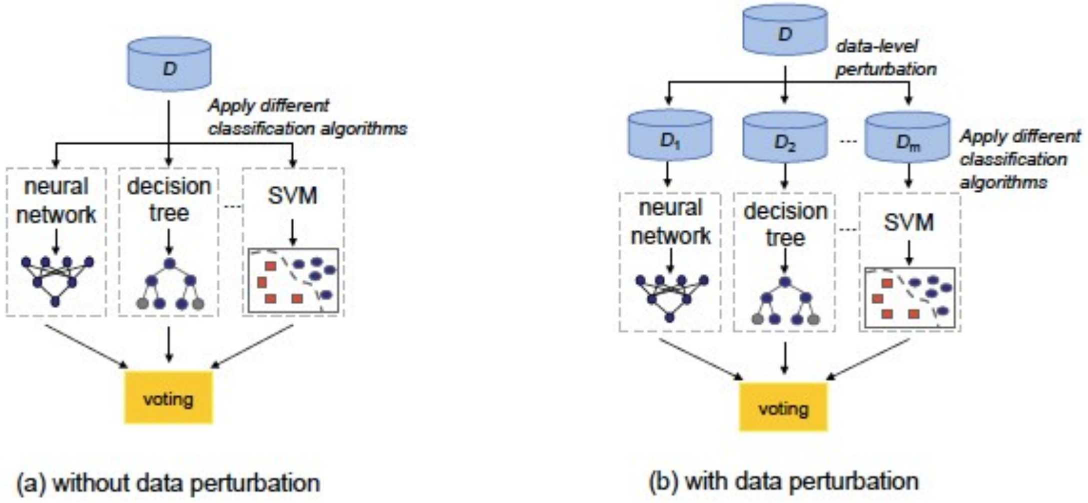

D
Apply different classification algorithms
neural network
decision tree
SVM
voting
(a) without data perturbation
D
data-level perturbation
D_1
D_2
D_m
Apply different classification algorithms
neural network
decision tree
SVM
voting
(b) with data perturbation

Figure 16 Schematic illustration of an ensemble using different classification

Source: Pengyi, Y., Yee, H.Y., Zhou, B.B. and Zomaya, A.Y. (2010)

Other approaches for the creation of ensembles are possible. Liu et al. (Liu et al., 2004) introduced a new ensemble of neural networks. It uses three different feature selection/extraction methods mixed with bootstrapping to provide diverse base classifiers.

They indicated that diverse base classifiers could also be achieved by including different feature generating algorithms that offer various gene ranking lists. Koziol et al. (Koziol et al., 2009) used the same concept similar. Amaratunga et al. (Amaratunga et al., 2008) created a random forests variant named enriched random forest. It weighs the importance of features when selecting splitting nodes. It enhanced the random forests algorithm for the very high-dimensional dataset. That modification on the random forests showed very encouraging results, especially when the dimension of the microarray data is significant and the number of the discriminative genes is small. Solution-aggregating motif finder was presented by Yanover et al. (Yanover et al., 2009). It is a statistical ensemble method. Their proposed method was based on Markov Random Field with the Best max-marginal first algorithm (Yanover et al., 2004), which provides the M top-scoring solutions. Armananzas et al. (Armananzas et al., 2008) suggested a hierarchy of Bayesian network classifiers to detect gene interactions from microarray data. Robles et al. (Robles et al., 2004) used a Bayesian network to combine multiple classifiers for the use of protein secondary structure prediction. Hu et al. (Hu et al., 2006) and Wijaya et al. (Wijaya et al., 2008) suggested combining the outputs of multiple motif finder algorithms to improve the final prediction result.

3.5 Summary

This chapter focused on providing an overview and description of machine learning algorithms often used in research and their applications. Machine learning has been used in data mining, computer vision, natural language processing, medical diagnostics, DNA sequence sequencing, and robotics. This chapter started with a description and applications of neural networks. It discussed instance-based learning algorithms such as KNN, kernel-based algorithms such as SVM and ensemble learning. Artificial Neural networks can be used to model complex relationships between inputs and outputs or to find patterns in data. The k-nearest neighbors (KNN) can be used to solve both classification and regression problems. It is one of the most used learning algorithms. It is based on feature similarity. SVM is one of the best choices for high-performance algorithms. Some use-cases of SVM include face detection, handwriting detection and bioinformatics. Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Due to growing computational power in recent years allowing to train larger ensemble learning in a short period, the number of applications of ensemble learning has grown increasingly. Some current applications include computer security such as classification of malware codes or anomalies detection or applications in remote sensing such as land cover mapping or change detection.

4. Datasets and data preparation

Several databases are available and are repositories for the structures and sequences of transmembrane proteins.

4.1 Existing datasets

Orientations of Proteins in Membranes (OPM) database (Lomize, Lomize, Pogozheva and Mosberg, 2006) include a list of transmembrane, peripheral and monotopic proteins extracted from the Protein Data Bank (PDB). With the database, it is possible to analyse or search membrane proteins based on various parameters such as structural classification, destination membrane or the numbers of TM segments, to name a few.

PDB_TM is a protein data bank of transmembrane proteins (Tusnady, Dosztanyi and Simon, 2005a). As of March 18, 2016, it contains 2745 transmembrane proteins (2394 alpha and 336 betas). The PDB_TM database was created by scanning all PDB entries with the TMDET algorithm (Tusnady, Dosztanyi and Simon, 2005b).

Membrane Protein Bioinformatics Research Group at the Institute of Enzymology.

CGDB is a database of membrane proteins/lipid interactions by coarse-grained molecular dynamics simulations. The lipid environment is a factor that affects membrane protein functions and stability. Chetwynd et al. developed a computation approach to predict membrane protein/lipid interactions (Chetwynd, Scott, Mokra and Sansom, 2008).

MPDB is a membrane protein database that contains various information concerning the structures and functions of membrane proteins and peptides. Data is derived from the PDB and other databases, and it includes integral, anchored, and peripheral membrane proteins and peptides. X-ray diffraction, nuclear magnetic resonance, electron diffractions and cryo-electron microscopy are used for the basis of the structures. (Raman, Cherezov and Caffrey, 2006). It contains 1095 unique entries (285 unique proteins/peptides and 155 unique protein families).

The team at Stephen White laboratory work mostly on biophysics issues concerning the folding and stability of membrane proteins (The Stephen White Laboratory at UC Irvine, 2016). They have few available resources, including the ‘Membrane Proteins of Known 3D Structure’.

Other databases available from the laboratory include the Membrane Protein Explorer (MPEx). MPEx is a tool that uses hydropathy plots to analyse the topology and features of membrane proteins.

MPtopo is also a curated database available as part of the Stephen White laboratory at UC Irvine. The verification of topologies is achieved using crystallography, gene fusion, antibody, and mutagenesis studies (Jayasinghe, Hristova and White, 2001). The server is accessible, and users can query sequences with an SQL-based search engine. MPtopo contains a total of 165 proteins with 949 transmembrane segments.

TOPDB (Topology data bank of Transmembrane Proteins) contains a comprehensive list of transmembrane proteins with topology information (Tusnady, Kalmár and Simon, 2008). It has 4190 transmembrane proteins obtained from the literature and public databases available on the internet. The database is available at: http://topdb.enzim.hu

OMPdb is a database that contains beta-barrel outer membrane proteins from Gram-negative bacteria. It is the most complete and comprehensive collection of integral beta-barrel outer membrane proteins (Tsirigos, Bagos and Hamodrakas, 2011). As of April 20, 2016, the database contains 372,536 proteins entries with 93 families based on the structural and functional criteria.

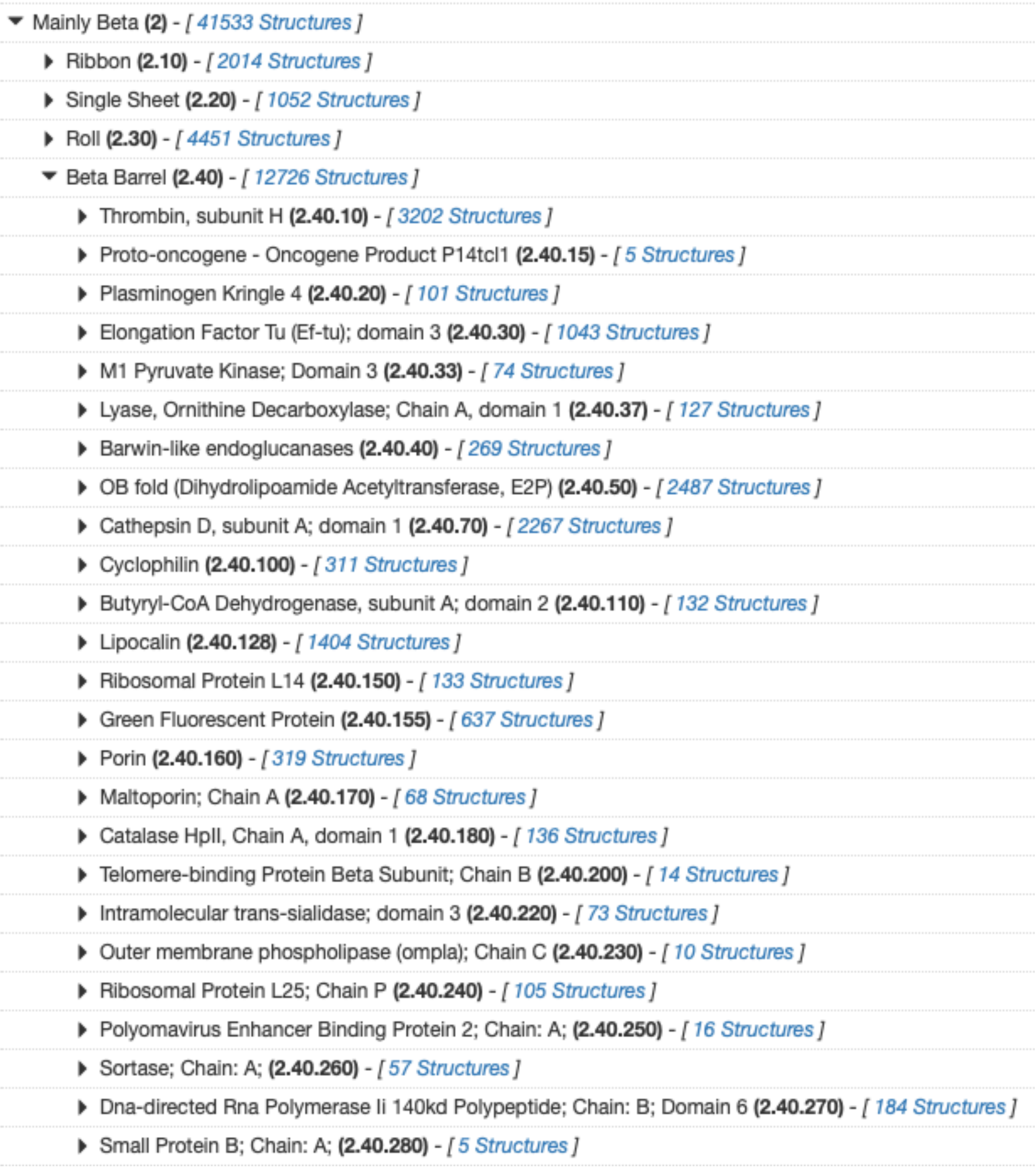
OMPdb (simulations) is a database of outer membrane protein simulations. It contains summaries from molecular dynamic simulations of around 20 transmembrane beta-barrel proteins. There are also indications of lipid contacts. Dr Syma Khalid currently maintains the database, and work is supported by Dr Kathryn Scott, Dr Peter J. Bond, and Anthony Ivetac.

TMBB-DB is a transmembrane beta-barrel database developed and maintained by the lab of William Wimley (Freeman and Wimley, 2012). It is an ensemble of the predictions obtained using the Freeman-Wimley algorithm, which was shown to be among the most accurate predictor of TMBBs. N-terminal export signal peptide predictions made by the SignalP server are also included.

TCDB (Transport Classification Database) is a curated, relational database containing sequence, classification, structural-functional and evolutionary information about transport systems from various living organisms (Saier, Tran and Barabote, 2006). It is possible to look for proteins that belong to a specific porin family. There are 81 different families.

UniProt Knowledgebase (UniProtKB) is a non-redundant database automatically annotated and manually curated by experts. It is possible to search for the origin of all data as the information is provided with the source. There are more than 120 external databases to which UnitProtKB is cross-referenced, and there are releases every four weeks. More than 68 million entries in UniProtKB as of release 2016_09 of October 5, 2016, with 552 259 entries in Swiss-Prot and 67 940 995 entries in TrEMBL. Records in Swiss-Prot have been manually annotated, and documents in TrEMBL are awaiting manual annotation.

The Protein Data Bank archive (PDB) has been used since 1971. It has much information regarding the 3D structures of proteins, nucleic acids, and complex assemblies. The Worldwide PDB is the organisation that makes sure that the PDB remains free and publicly available. TM proteins available in the PDB are identified using the mpstruc database of Stephen White, UC Irvin, sequence clustering and data derived from UniProt. A membrane protein browser is available, and as of 12 Novembre 2021, the data bank contains 12726 beta-barrel structures. An example of structures found in PDB is represented in figure 17.

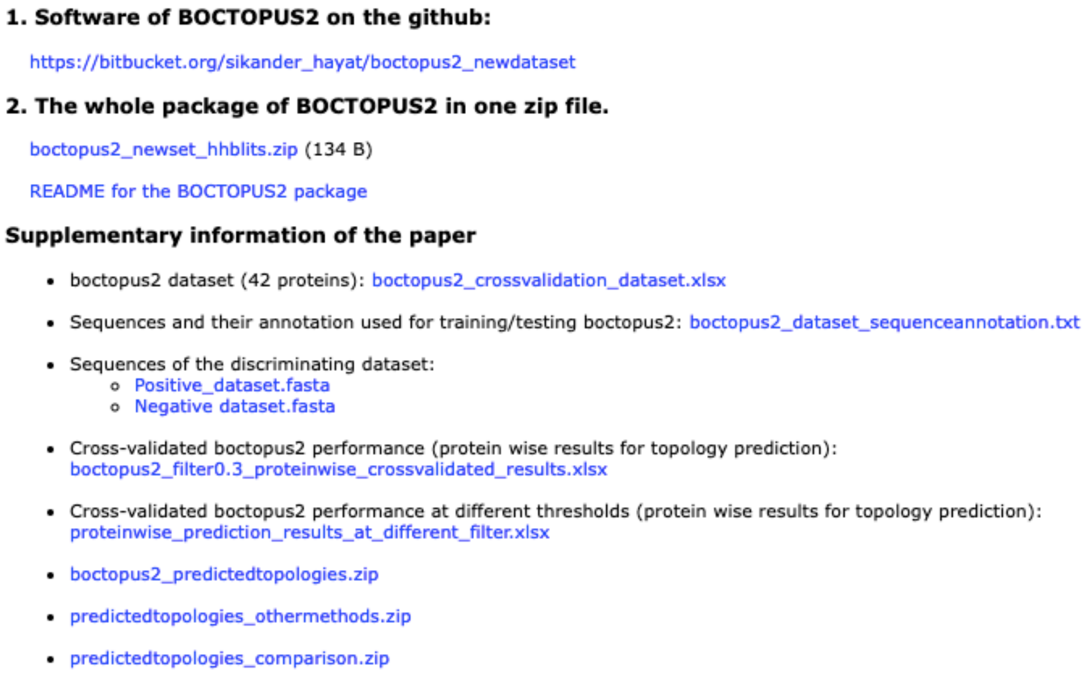


- ▼ Mainly Beta **(2)** - *[41533 Structures]*
  - ▶ Ribbon **(2.10)** - *[2014 Structures]*
  - ▶ Single Sheet **(2.20)** - *[1052 Structures]*
  - ▶ Roll **(2.30)** - *[4451 Structures]*
  - ▼ Beta Barrel **(2.40)** - *[12726 Structures]*
    - ▶ Thrombin, subunit H **(2.40.10)** - *[3202 Structures]*
    - ▶ Proto-oncogene - Oncogene Product P14tcl1 **(2.40.15)** - *[5 Structures]*
    - ▶ Plasminogen Kringle 4 **(2.40.20)** - *[101 Structures]*
    - ▶ Elongation Factor Tu (Ef-tu); domain 3 **(2.40.30)** - *[1043 Structures]*
    - ▶ M1 Pyruvate Kinase; Domain 3 **(2.40.33)** - *[74 Structures]*
    - ▶ Lyase, Ornithine Decarboxylase; Chain A, domain 1 **(2.40.37)** - *[127 Structures]*
    - ▶ Barwin-like endoglucanases **(2.40.40)** - *[269 Structures]*
    - ▶ OB fold (Dihydrolipoamide Acetyltransferase, E2P) **(2.40.50)** - *[2487 Structures]*
    - ▶ Cathepsin D, subunit A; domain 1 **(2.40.70)** - *[2267 Structures]*
    - ▶ Cyclophilin **(2.40.100)** - *[311 Structures]*
    - ▶ Butyryl-CoA Dehydrogenase, subunit A; domain 2 **(2.40.110)** - *[132 Structures]*
    - ▶ Lipocalin **(2.40.128)** - *[1404 Structures]*
    - ▶ Ribosomal Protein L14 **(2.40.150)** - *[133 Structures]*
    - ▶ Green Fluorescent Protein **(2.40.155)** - *[637 Structures]*
    - ▶ Porin **(2.40.160)** - *[319 Structures]*
    - ▶ Maltoporin; Chain A **(2.40.170)** - *[68 Structures]*
    - ▶ Catalase HpII, Chain A, domain 1 **(2.40.180)** - *[136 Structures]*
    - ▶ Telomere-binding Protein Beta Subunit; Chain B **(2.40.200)** - *[14 Structures]*
    - ▶ Intramolecular trans-sialidase; domain 3 **(2.40.220)** - *[73 Structures]*
    - ▶ Outer membrane phospholipase (ompla); Chain C **(2.40.230)** - *[10 Structures]*
    - ▶ Ribosomal Protein L25; Chain P **(2.40.240)** - *[105 Structures]*
    - ▶ Polyomavirus Enhancer Binding Protein 2; Chain: A; **(2.40.250)** - *[16 Structures]*
    - ▶ Sortase; Chain: A; **(2.40.260)** - *[57 Structures]*
    - ▶ Dna-directed Rna Polymerase Ii 140kd Polypeptide; Chain: B; Domain 6 **(2.40.270)** - *[184 Structures]*
    - ▶ Small Protein B; Chain: A; **(2.40.280)** - *[5 Structures]*

Figure 17 RCSB-PDB hierarchical classification of protein domain structures
Source: RCSB Protein Data Bank (2021)

4.2 BOCTOPUS2 dataset

The BOCTOPUS2 dataset has been used and implemented in MATLAB. The dataset was used for the training and testing of the BOCTOPUS2 tool that is used for TMB topology prediction (Hayat, Peters, Shu, Tsirigos and Elofsson, 2016). It is available as supplementary information of the paper and the URL for the BOCTOPUS2 software.



1. Software of BOCTOPUS2 on the github:

https://bitbucket.org/sikander_hayat/boctopus2_newdataset

2. The whole package of BOCTOPUS2 in one zip file.

boctopus2_newset_hhblits.zip (134 B)

README for the BOCTOPUS2 package

Supplementary information of the paper

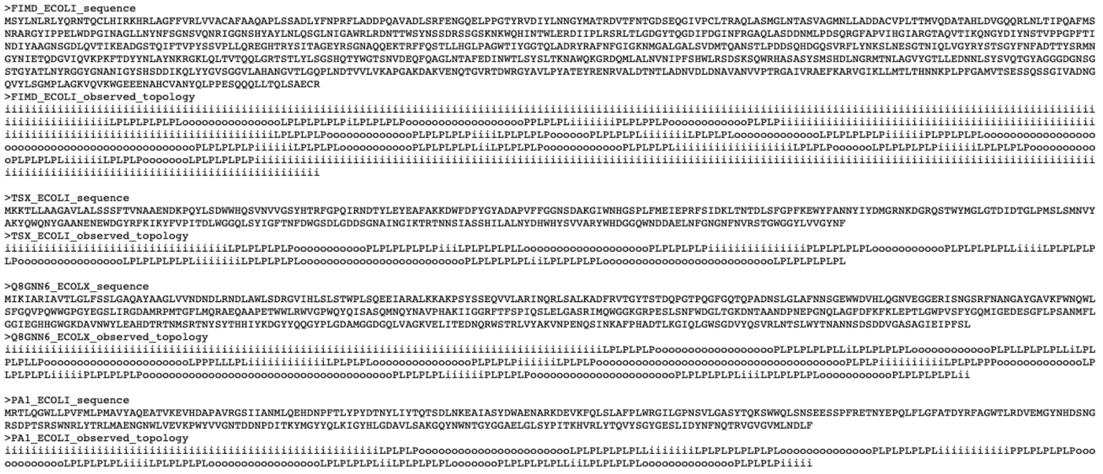
- boctopus2 dataset (42 proteins): boctopus2_crossvalidation_dataset.xlsx
- Sequences and their annotation used for training/testing boctopus2: boctopus2_dataset_sequenceannotation.txt
- Sequences of the discriminating dataset:
  - Positive_dataset.fasta
  - Negative dataset.fasta
- Cross-validated boctopus2 performance (protein wise results for topology prediction): boctopus2_filter0.3_proteinwise_crossvalidated_results.xlsx
- Cross-validated boctopus2 performance at different thresholds (protein wise results for topology prediction): proteinwise_prediction_results_at_different_filter.xlsx
- boctopus2_predictedtopologies.zip
- predictedtopologies_othermethods.zip
- predictedtopologies_comparison.zip

Figure 18 BOCTOPUS 2: Improved topology prediction of transmembrane beta-barrel proteins

Source: Topcons.net (2021)

The BOCTOPUS2 dataset consists of 42 TMB sequences represented in table 1.TMB proteins are identified by their corresponding crossvalidation_setid, protein data bank ID (PDB_ID), UniProt ID (UNIPROT_ID), superfamily classes(superfamily) and barrel region (barrel_region). TMB proteins include nucleoside-specific channel-forming Tsx (UniProt ID: TSX_ECOLI, PDB ID:1TLY_A), which is part of the mpstruc database represented in figure 19. The 42 TMB sequences are culled at 30% sequence identity using the PISCES server (Wang and Dunbrack, 2003) and are divided into ten subsets for cross-validation.

To ensure no homology information is used, all proteins belonging to the same super-family are put into the same cross-validation group. Membrane boundaries and super-family classification are obtained from the OPM database (Lomize, Lomize, Pogozheva and Mosberg, 2006). Residues in the transmembrane strands are labelled as pore-facing (p) and lipid-facing(l) based on their side-chain orientation relative to the barrel centre. This can be observed in the file boctopus2_dataset_sequenceannotation.txt. In the file, (o) corresponds to outer-loop and (i) corresponds to inner-loop.



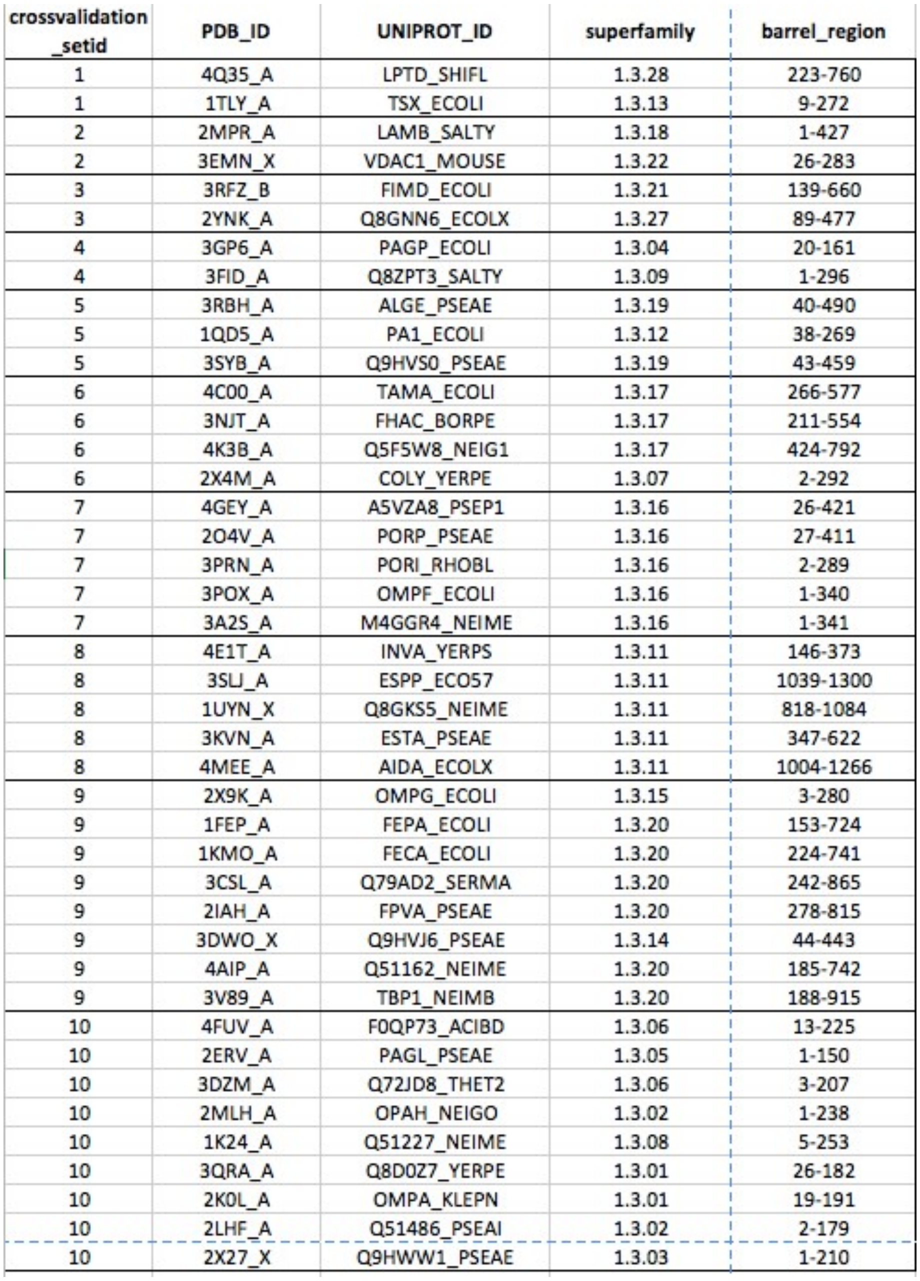
```
>FIMD_ECOLI_sequence
MSYLNLRLYQRNTQCLHIRKHRLAGFFVRLVVACAFAAQAPLSSADLYFNPRFLADDPQAVADLSRFENGQELPPGTYRVDIYLNNGYMATRDVTFNTGDSEQGIVPCLTRAQLASMGLNTASVAGMNLLADDACVPLTTMVQDATAHLDVGQQRLNLTIPQAFMS
NRARGYIPPELWDPGINAGLLNYNFSGNSVQNRIGGNSHYAYLNLQSGLNIGAWRLRDNTTWSYNSSDRSSGSKNKWQHINTWLERDIIPLRSRLTLGDYTQGDIFDGINFRGAQLASDDNMLPDSQRGFAPVIHGIARGTAQVTIKQNGYDIYNSTVPPGPFTI
NDIYAAGNSGDLQVTIKEADGSTQIFTVPYSSVPLLQREGHTRYSITAGEYRSGNAQQEKTRFFQSTLLHGLPAGWTIYGGTQLADRYRAFNFGIGKNMGALGALSVDMTQANSTLPDDSQHDGQSVRFLYNKSLNESGTNIQLVGYRYSTSGYFNFADTTYSRMN
GYNIETQDGVIQVKPKFTDYYNLAYNKRGKLQLTVTQQLGRTSTLYLSGSHQTYWGTSNVDEQFQAGLNTAFEDINWTLSYSLTKNAWQKGRDQMLALNVNIPFSHWLRSDSKSQWRHASASYSMSHDLNGRMTNLAGVYGTLLEDNNLSYSVQTGYAGGGDGNSG
STGYATLNYRGGYGNANIGYSHSDDIKQLYYGVSGGVLAHANGVTLGQPLNDTVVLVKAPGAKDAKVENQTGVRTDWRGYAVLPYATEYRENRVALDTNTLADNVDLDNAVANVVPTRGAIVRAEFKARVGIKLLMTLTHNNKPLPFGAMVTSESSQSSGIVADNG
QVYLSGMPLAGKVQVKWGEEENAHCVANYQLPPESQQQLLTQLSAECR
>FIMD_ECOLI_observed_topology
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
iiiiiiiiiiiiiiiiLPLPLPLPLPLoooooooooooooooLPLPLPLPLPiLPLPLPLPooooooooooooooooooPPLPLPLiiiiiiiPLPLPPLPoooooooooooooPLPLPiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLPLPLPLPoooooooooooooPLPLPLPLPiiiiLPLPLPLPooooooPLPLPLPLiiiiiiiLPLPLPLoooooooooooooLPLPLPLPLPiiiiiiPLPPLPLPLoooooooooooooooo
oooooooooooooooooooooooooooooPLPLPLPLPiiiiiiLPLPLPLoooooooooooPLPLPLPLPLiiLPLPLPLPoooooooooooPLPLPLPLiiiiiiiiiiiiiiiiiiiLPLPLPooooooLPLPLPLPLPiiiiiLPLPLPLPoooooooooo
oPLPLPLPLiiiiiiLPLPLPoooooooLPLPLPLPLPiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii

>TSX_ECOLI_sequence
MKKTLLAAGAVLALSSSFTVNAAENDKPQYLSDWWHQSVNVVGSYHTRFGPQIRNDTYLEYEAFAKKDWFDFYGYADAPVFFGGNSDAKGIWNHGSPLFMEIEPRFSIDKLTNTDLSFGPFKEWYFANNYIYDMGRNKDGRQSTWYMGLGTDIDTGLPMSLSMNVY
AKYQWQNYGAANENEWDGYRFKIKYFVPITDLWGGQLSYIGFTNFDWGSDLGDDSGNAINGIKTRTNNSIASSHILALNYDHWHYSVVARYWHDGGQWNDDAELNFGNGNFNVRSTGWGGYLVVGYNF
>TSX_ECOLI_observed_topology
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLPLPLPLPLPoooooooooooPLPLPLPLPLPiiiLPLPLPLPLLoooooooooooooooooooooPLPLPLPLPiiiiiiiiiiiiiiiPLPLPLPLPLooooooooooooPLPLPLPLPLLiiiiLPLPLPLP
LPoooooooooooooooooLPLPLPLPLPLiiiiiiiLPLPLPLPLoooooooooooooooooooooooooooPLPLPLPLPLiiLPLPLPLPLoooooooooooooooooooooooooooLPLPLPLPLPL

>Q8GNN6_ECOLX_sequence
MIKIARIAVTLGLFSSLGAQAYAAGLVVNDNDLRNDLAWLSDRGVIHLSLSTWPLSQEEIARALKKAKPSYSSEQVVLARINQRLSALKADFRVTGYTSTDQPGTPQGFGQTQPADNSLGLAFNNSGEWWDVHLQGNVEGGERISNGSRFNANGAYGAVKFWNQWL
SFGQVPQWWGPGYEGSLIRGDAMRPMTGFLMQRAEQAAPETWWLRWVGPWQYQISASQMNQYNAVPHAKIIGGRFTFSPIQSLELGASRIMQWGGKGRPESLSNFWDGLTGKDNTAANDPNEPGNQLAGFDFKFKLEPTLGWPVSFYGQMIGEDESGFLPSANMFL
GGIEGHHGWGKDAVNWYLEAHDTRTNMSRTNYSYTHHIYKDGYYQQGYPLGDAMGGDGQLVAGKVELITEDNQRWSTRLVYAKVNPENQSINKAFPHADTLKGIQLGWSGDVYQSVRLNTSLWYTNANNSDSDDVGASAGIEIPFSL
>Q8GNN6_ECOLX_observed_topology
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLPLPLPLPooooooooooooooooooPLPLPLPLPLLiLPLPLPLPLooooooooooooPLPLLPLPLPLLiLPL
PLPLLPoooooooooooooooooooooLPPPLLLPLiiiiiiiiiiiiLPLPLPLoooooooooooooooPLPLPLPiiiiiiLPLPLPooooooooooooooooooooooooooooooooooooooPLPLPiiiiiiiiiiiLPLPLPPPoooooooooooooLP
LPLPLPLiiiiiPLPLPLPLPooooooooooooooooooooooooooooooooooooooPLPLPLPLiiiiiPLPLPLPLPooooooooooooooooooooooooPLPLPLPLPLiiiLPLPLPLPLooooooooooooPLPLPLPLPLii

>PA1_ECOLI_sequence
MRTLQGWLLPVFMLPMAVYAQEATVKEVHDAPAVRGSIIANMLQEHDNPFTLYPYDTNYLIYTQTSDLNKEAIASYDWAENARKDEVKFQLSLAFPLWRGILGPNSVLGASYTQKSWWQLSNSEESSPFRETNYEPQLFLGFATDYRFAGWTLRDVEMGYNHDSNG
RSDPTSRSWNRLYTRLMAENGNWLVEVKPWYVVGNTDDNPDITKYMGYYQLKIGYHLGDAVLSAKGQYNWNTGYGGAELGLSYPITKHVRLYTQVYSGYGESLIDYNFNQTRVGVGVMLNDLF
>PA1_ECOLI_observed_topology
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLPLPLPoooooooooooooooooooooooLPLPLPLPLPLLiiiiiiiLPLPLPLPLPLPLooooooooooooooPLPLPLPLPLiiiiiiiiiiiPPLPLPLPLPooo
oooooooooLPLPLPLPLiiiiLPLPLPLPLooooooooooooooooooLPLPLPLPLiiLPLPLPLPLoooooooPLPLPLPLPLLiiLPLPLPLPLooooooooooooooPLPLPLPiiiii
```

Figure 19 boctopus2_dataset_sequenceannotation.txt

Source: Topcons.net (2021)

Table 1 BOCTOPUS2 dataset

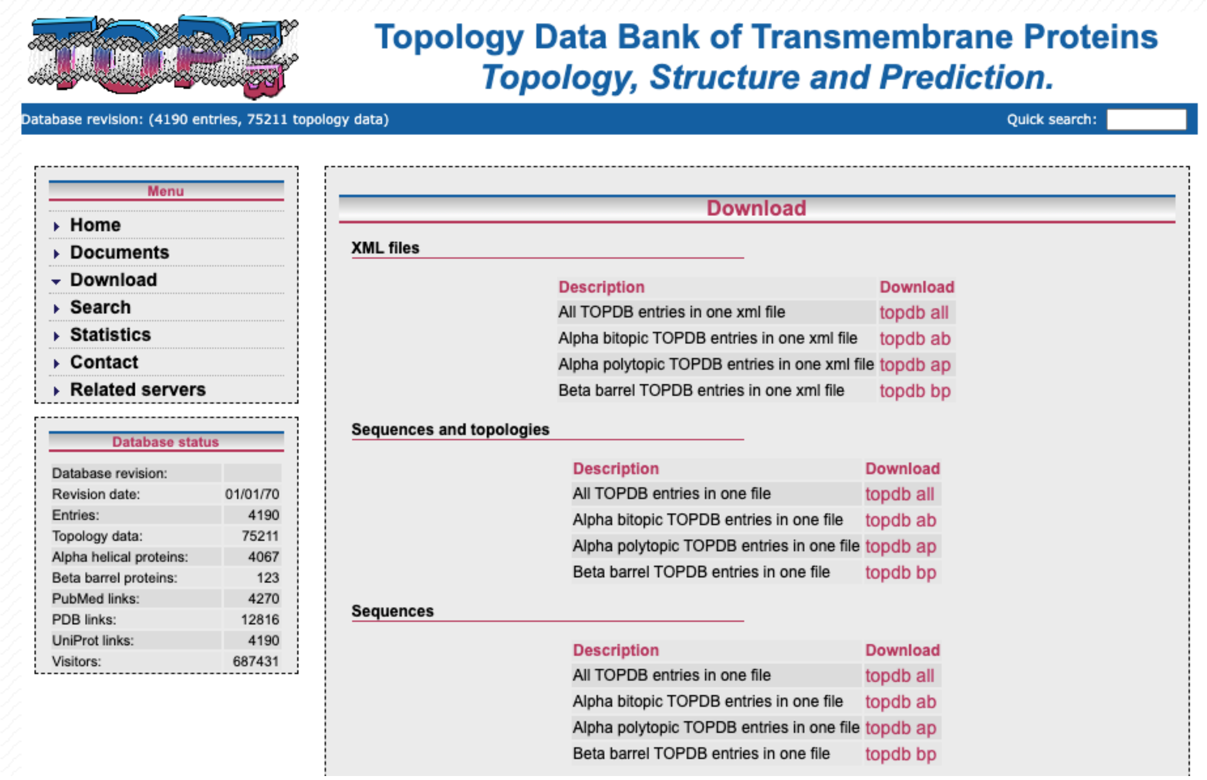
Source: Topcons.net (2021)



crossvalidation _setid	PDB_ID	UNIPROT_ID	superfamily	barrel_region
1	4Q35_A	LPTD_SHIFL	1.3.28	223-760
1	1TLY_A	TSX_ECOLI	1.3.13	9-272
2	2MPR_A	LAMB_SALTY	1.3.18	1-427
2	3EMN_X	VDAC1_MOUSE	1.3.22	26-283
3	3RFZ_B	FIMD_ECOLI	1.3.21	139-660
3	2YNK_A	Q8GNN6_ECOLX	1.3.27	89-477
4	3GP6_A	PAGP_ECOLI	1.3.04	20-161
4	3FID_A	Q8ZPT3_SALTY	1.3.09	1-296
5	3RBH_A	ALGE_PSEAE	1.3.19	40-490
5	1QD5_A	PA1_ECOLI	1.3.12	38-269
5	3SYB_A	Q9HVS0_PSEAE	1.3.19	43-459
6	4C00_A	TAMA_ECOLI	1.3.17	266-577
6	3NJT_A	FHAC_BORPE	1.3.17	211-554
6	4K3B_A	Q5F5W8_NEIG1	1.3.17	424-792
6	2X4M_A	COLY_YERPE	1.3.07	2-292
7	4GEY_A	A5VZA8_PSEP1	1.3.16	26-421
7	2O4V_A	PORP_PSEAE	1.3.16	27-411
7	3PRN_A	PORI_RHOBL	1.3.16	2-289
7	3POX_A	OMPF_ECOLI	1.3.16	1-340
7	3A2S_A	M4GGR4_NEIME	1.3.16	1-341
8	4E1T_A	INVA_YERPS	1.3.11	146-373
8	3SLJ_A	ESPP_ECO57	1.3.11	1039-1300
8	1UYN_X	Q8GKS5_NEIME	1.3.11	818-1084
8	3KVN_A	ESTA_PSEAE	1.3.11	347-622
8	4MEE_A	AIDA_ECOLX	1.3.11	1004-1266
9	2X9K_A	OMPG_ECOLI	1.3.15	3-280
9	1FEP_A	FEPA_ECOLI	1.3.20	153-724
9	1KMO_A	FECA_ECOLI	1.3.20	224-741
9	3CSL_A	Q79AD2_SERMA	1.3.20	242-865
9	2IAH_A	FPVA_PSEAE	1.3.20	278-815
9	3DWO_X	Q9HVJ6_PSEAE	1.3.14	44-443
9	4AIP_A	Q51162_NEIME	1.3.20	185-742
9	3V89_A	TBP1_NEIMB	1.3.20	188-915
10	4FUV_A	F0QP73_ACIBD	1.3.06	13-225
10	2ERV_A	PAGL_PSEAE	1.3.05	1-150
10	3DZM_A	Q72JD8_THET2	1.3.06	3-207
10	2MLH_A	OPAH_NEIGO	1.3.02	1-238
10	1K24_A	Q51227_NEIME	1.3.08	5-253
10	3QRA_A	Q8D0Z7_YERPE	1.3.01	26-182
10	2K0L_A	OMPA_KLEPN	1.3.01	19-191
10	2LHF_A	Q51486_PSEAI	1.3.02	2-179
10	2X27_X	Q9HWW1_PSEAE	1.3.03	1-210

4.3 TOPDB dataset

The TOPDB dataset will also be used for the implementation after trying the BOCTOPUS2 dataset. TOPDB entries are available at the Topology Data Bank of Transmembrane Proteins web server.



Topology Data Bank of Transmembrane Proteins
Topology, Structure and Prediction.

Database revision: (4190 entries, 75211 topology data)

Database status

Database revision:	
Revision date:	01/01/70
Entries:	4190
Topology data:	75211
Alpha helical proteins:	4067
Beta barrel proteins:	123
PubMed links:	4270
PDB links:	12816
UniProt links:	4190
Visitors:	687431

Download

XML files

Description	Download
All TOPDB entries in one xml file	topdb all
Alpha bitopic TOPDB entries in one xml file	topdb ab
Alpha polytopic TOPDB entries in one xml file	topdb ap
Beta barrel TOPDB entries in one xml file	topdb bp

Sequences and topologies

Description	Download
All TOPDB entries in one file	topdb all
Alpha bitopic TOPDB entries in one file	topdb ab
Alpha polytopic TOPDB entries in one file	topdb ap
Beta barrel TOPDB entries in one file	topdb bp

Sequences

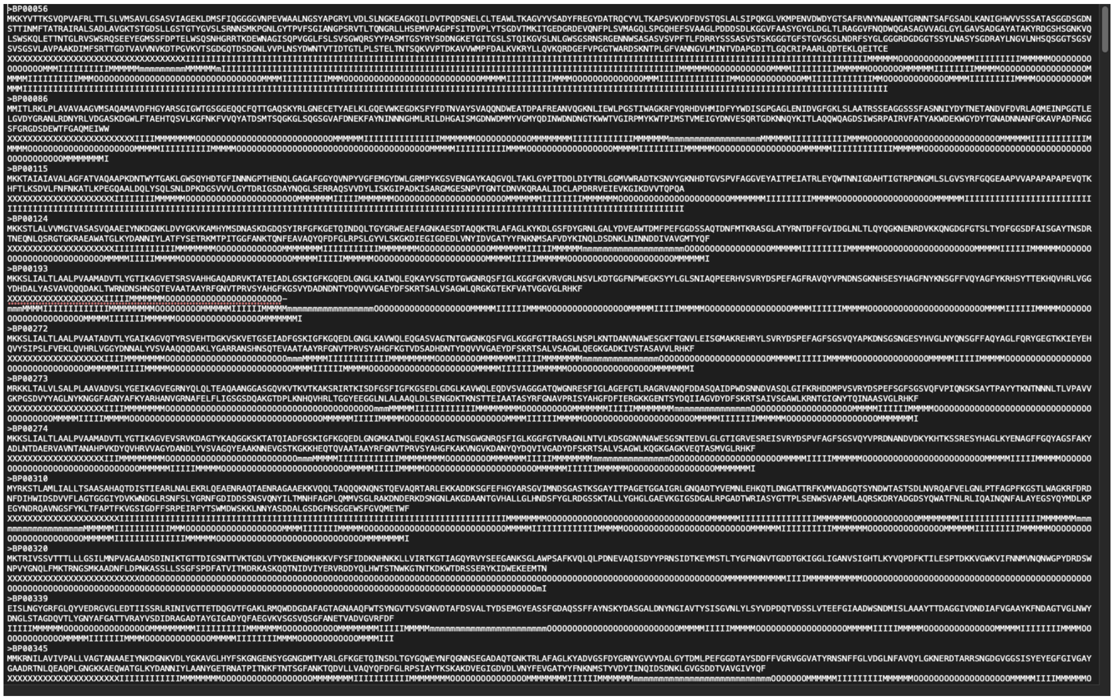
Description	Download
All TOPDB entries in one file	topdb all
Alpha bitopic TOPDB entries in one file	topdb ab
Alpha polytopic TOPDB entries in one file	topdb ap
Beta barrel TOPDB entries in one file	topdb bp

Figure 20 Topology Data Bank of Transmembrane Proteins Topology, Structure and Prediction

Source: Enzim. hu. (2014)

The beta-barrel TOPDB entries can be downloaded directly from the website and are available at: http://topdb.enzim.hu/?m=download&mid=2. The topdb_bp.txt file contains 123 TMB sequences larger than the BOCTOPUS2 dataset.

A representation of the file’s content is found in the following picture.



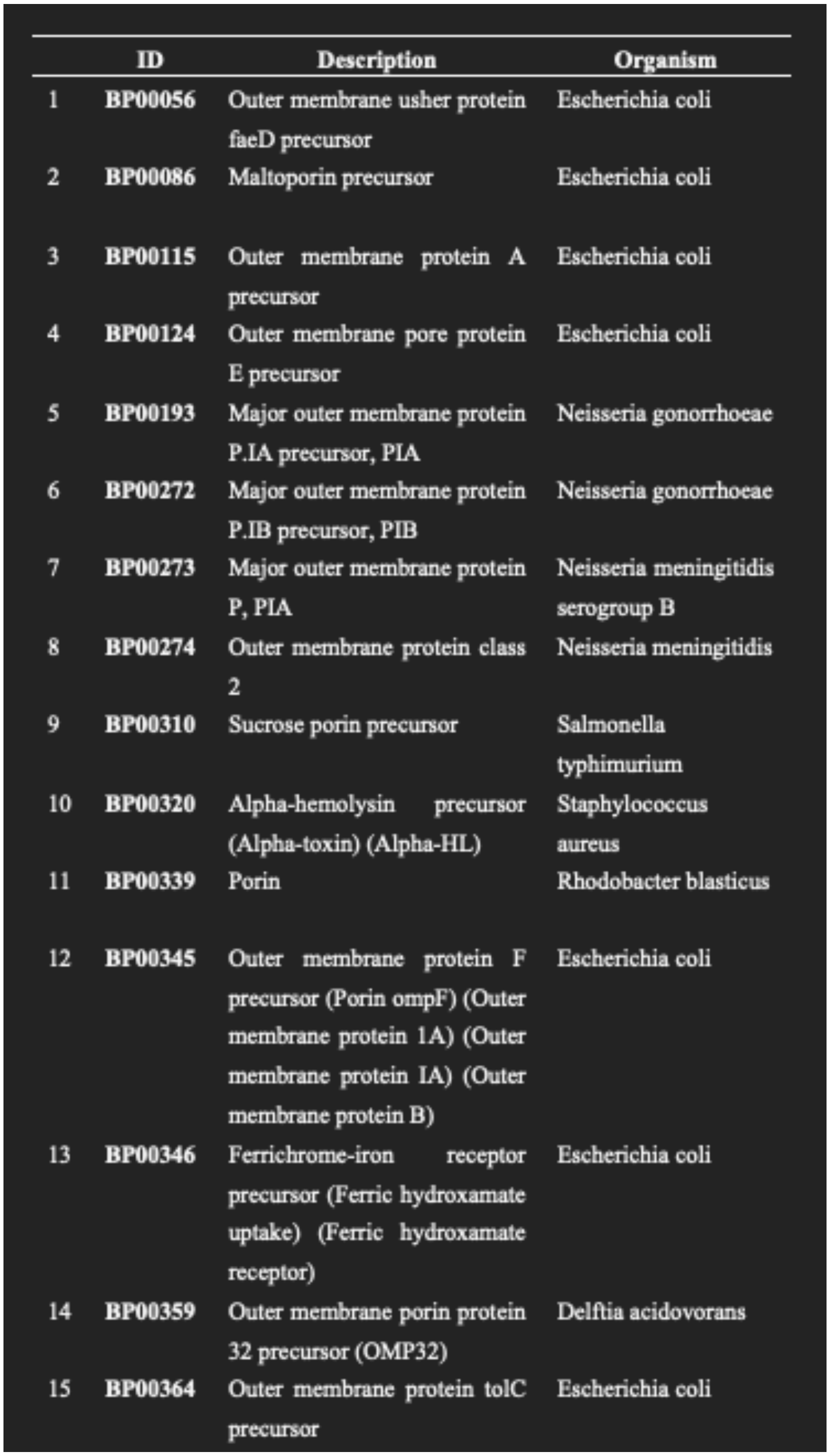
```
>BP00056
MKKYVTTKSVQPVAFRLTTLSLVMSAVLGSASVIAGEKLDMSFIQGGGGVNPEVWAALNGSYAPGRYLVDLSLNGKEAGKQILDVTPQDSNELCLTEAWLTKAGVYVSADYFREGYDATRQCYVLTKAPSVKVDFDVSTQSLALSIPQKGLVKMPENVDWDYGTSAFRVNYNANANTGRNNTSAFGSADLKANIGHWVVSSSATASGGDSGDN
STTINMFTATRAIRALSADLAVGKTSTGDSLLGSTGTYGVSLSRNNSMKPGNLGYTPVFSGIANGPSRVTLTQNGRLLHSEMVPAGPFSITDVPLYTSGDVTMKITGEDGRDEVQNFPLSVMAGQLSPGQHEFSVAAGLPDDDSDLKGGVFAASYGYGLDGLTLRAGGVFNQDWQGASAGVVAGLGYLGAVSADGAYATAKYRDGSHSGNKVQ
LSWSKQLETTNTGLRVSWSRQSEEYEGMSSFDPTELWSQSNHGRRTKDEWNAGISQPVGGLFSLSVSGWQRSYYPASMTGSYRYSDDNGKETGITGSLSTQIKGVSLNLGWSGSRNSRGENNWSASASVSVPFTLFDRRYSSSASVSTSKGGGTGFSTGVSGSLNDRFSYGLGGGRDGDGGTSSYLNASYSGDRAYLNGVLNHSQSGGTSGSV
SVSGSVLAVPAAKDIMFSRTTGDTVAVVNVKDTPGVKVTSGDGQTDSDGNLVVPLNSYDWNTVTIDTGTLPLSTELTNTSQKVVPTDKAVVWMPFDALKVKRYLLQVKQRDGEFVPGGTWARDSKNTPLGFVANNGVLMINTVDAPGDITLGQCRIPAARLQDTEKLQEITCE
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMOOOOOOOOOOOOMMMMIIIIIIIIIIIMMMMMMOOOOOOOO
OOOOOOOMMMIIIIIIIIIIIMMMMMMmmmmmmmmmmMMMMMMmIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMOOOOOOOOOOOOMMMMIIIIIIIIIIMMMMMMOOOOOOOMMMMMIIIIIIIIMMMMMOOOOOOOOOOOOOOOOOOM
MMMMIIIIIIIIIIIMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMIIIIIIIIIIMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIIIIIMMMMOOOOOOOOOOOOOOOOMMMIIIIIIIIIIIIIMMOOOOOOOOOOOOOMMIIIIIIIIIIIIMMOOOOOOOOOOOOOOMMMMIIIIIIIIIIMMMMOOOOOOOOOOOM
MMMIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
>BP00086
MMITLRKLPLAVAVAAGVMSAQAMAVDFHGYARSGIGWTGSGGEQQCFQTTGAQSKYRLGNECETYAELKLGQEVWKEGDKSFYFDTNVAYSVAQQNDWEATDPAFREANVQGKNLIEWLPGSTIWAGKRFYQRHDVHMIDFYYWDISGPGAGLENIDVGFGKLSLAATRSSEAGGSSSFASNNIYDYTNETANDVFDVRLAQMEINPGGTLE
LGVDYGRANLRDNYRLVDGASKDGWLFTAEHTQSVLKGFNKFVVQYATDSMTSQGKGLSQGSGVAFDNEKFAYNINNNGHMLRILDHGAISMGDNWDMMYVGMYQDINWDNDNGTKWWTVGIRPMYKWTPIMSTVMEIGYDNVESQRTGDKNNQYKITLAQQWQAGDSIWSRPAIRVFATYAKWDEKWGYDYTGNADNNANFGKAVPADFNGG
SFGRGDSDEWTFGAQMEIWW
XXXXXXXXXXXXXXXXXXXXXXXXXIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMIIIIIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOMMMMMMMMIIIIIIIIIIIIMMMMMMMmmmmmmmmmmmmmmmmmmmMMMMMMIIIIIIIIIIIIMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMIIIIIIIIIIIIM
MMMMOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIIIIIIMMMMOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOMMMMMMIIIIIIIIIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOMMMMMMMMI
>BP00115
MKKTAIAIAVALAGFATVAQAAPKDNTWYTGAKLGWSQYHDTGFINNNGPTHENQLGAGAFGGYQVNPYVGFEMGYDWLGRMPYKGSVENGAYKAQGVQLTAKLGYPITDDLDIYTRLGGMVWRADTKSNVYGKNHDTGVSPVFAGGVEYAITPEIATRLEYQWTNNIGDAHTIGTRPDNGMLSLGVSYRFGQGEAAPVVAPAPAPAPEVQTK
HFTLKSDVLFNFNKATLKPEGQAALDQLYSQLSNLDPKDGSVVVLGYTDRIGSDAYNQGLSERRAQSVVDYLISKGIPADKISARGMGESNPVTGNTCDNVKQRAALIDCLAPDRRVEIEVKGIKDVVTQPQA
XXXXXXXXXXXXXXXXXXXXXIIIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMIIIIMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMIIIIIIIIMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMIIIIIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOMMMMMMIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
>BP00124
MKKSTLALVVMGIVASASVQAAEIYNKDGNKLDVYGKVKAMHYMSDNASKDGDQSYIRFGFKGETQINDQLTGYGRWEAEFAGNKAESDTAQQKTRLAFAGLKYKDLGSFDYGRNLGALYDVEAWTDMFPEFGGDSSAQTDNFMTKRASGLATYRNTDFFGVIDGLNLTLQYQGKNENRDVKKQNGDGFGTSLTYDFGGSDFAISGAYTNSDR
TNEQNLQSRGTGKRAEAWATGLKYDANNIYLATFYSETRKMTPITGGFANKTQNFEAVAQYQFDFGLRPSLGYVLSKGKDIEGIGDEDLVNYIDVGATYYFNKNMSAFVDYKINQLDSDNKLNINNDDIVAVGMTYQF
XXXXXXXXXXXXXXXXXXXXIIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOOOOMMMMMMMIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOOOOMMMMMMMIIIIIMMMMMMmmmmmmmmmmmmmmmmmmmmmOOOOOOOOOOOOOOOOOMMMMMIIIIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIMMMMMMOOOOOO
OOOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMI
>BP00193
MKKSLIALTLAALPVAAMADVTLYGTIKAGVETSRSVAHHGAQADRVKTATEIADLGSKIGFKGQEDLGNGLKAIWQLEQKAYVSGTDTGWGNRQSFIGLKGGFGKVRVGRLNSVLKDTGGFNPWEGKSYYLGLSNIAQPEERHVSVRYDSPEFAGFRAVQYVPNDNSGKNHSESYHAGFNYKNSGFFVQYAGFYKRHSYTTEKHQVHRLVGG
YDHDALYASVAVQQQDAKLTWRNDNSHNSQTEVAATAAYRFGNVTPRVSYAHGFKGSVYDADNDNTYDQVVVGAEYDFSKRTSALVSAGWLQRGKGTEKFVATVGGVGLRHKF
XXXXXXXXXXXXXXXXXXXIIIIIMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOO-
mmmMMMMIIIIIIIIIIIIIMMMMMMMMMOOOOOOOOOOMMMMMMIIIIIIMMMMMmmmmmmmmmmmmmmmmmmOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIMMMMOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMOOOOOOOOOOOOOOOOOOMMMMIIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOMMMMIIIIIIMMMMMOOOOOO
OOOOOOOOOOOOOOOMMMMMIIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMMMI
>BP00272
MKKSLIALTLAALPVAATADVTLYGAIKAGVQTYRSVEHTDGKVSKVETGSEIADFGSKIGFKGQEDLGNGLKAVWQLEQGASVAGTNTGWGNKQSFVGLKGGFGTIRAGSLNSPLKNTDANVNAWESGKFTGNVLEISGMAKREHRYLSVRYDSPEFAGFSGSVQYAPKDNSGSNGESYHVGLNYQNSGFFAQYAGLFQRYGEGTKKIEYEH
QVYSIPSLFVEKLQVHRLVGGYDNNALYVSVAAQQQDAKLYGARRANSHNSQTEVAATAAYRFGNVTPRVSYAHGFKGTVDSADHDNTYDQVVVGAEYDFSKRTSALVSAGWLQEGKGADKIVSTASAVVLRHKF
XXXXXXXXXXXXXXXXXXIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOmmmMMMMMIIIIIIIIIIIIIMMMMMMMMMMOOOOOOOOOOMMMMMMMIIIIIMMMMMMMIIIIIMMMMMMMMmmmmmmmmmmmmmmmOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIMMMMMOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOMMMMMMIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMIIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMMMI
>BP00273
MRKKLTALVLSALPLAAVADVSLYGEIKAGVEGRNYQLQLTEAQAANGGASGQVKVTKVTKAKSRIRTKISDFGSFIGFKGSEDLGDGLKAVWQLEQDVSVAGGGATQWGNRESFIGLAGEFGTLRAGRVANQFDDASQAIDPWDSNNDVASQLGIFKRHDDMPVSVRYDSPEFSGFSGSVQFVPIQNSKSAYTPAYYTKNTNNNLTLVPAVV
GKPGSDVYYAGLNYKNGGFAGNYAFKYARHANVGRNAFELFLIGSGSDQAKGTDPLKNHQVHRLTGGYEEGGLNLALAAQLDLSENGDKTKNSTTEIAATASYRFGNAVPRISYAHGFDFIERGKKGENTSYDQIIAGVDYDFSKRTSAIVSGAWLKRNTGIGNYTQINAASVGLRHKF
XXXXXXXXXXXXXXXXXXIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOmmmMMMMMIIIIIIIIIIIIMMMMMMMMMMOOOOOOOOOOMMMMMMIIIIIMMMMMMMMmmmmmmmmmmmmmmmOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOMMMMMMIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOMMMMIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMMMI
>BP00274
MKKSLIALTLAALPVAAMADVTLYGTIKAGVEVSRVKDAGTYKAQGGKSKTATQIADFGSKIGFKGQEDLGNGMKAIWQLEQKASIAGTNSGWGNRQSFIGLKGGFGTVRAGNLNTVLKDSGDNVNAWESGSNTEDVLGLGTIGRVESREISVRYDSPVFAGFSGSVQYVPRDNANDVDKYKHTKSSRESYHAGLKYENAGFFGQYAGSFAKY
ADLNTDAERVAVNTANAHPVKDYQVHRVVAGYDANDLYVSVAGQYEAAKNNEVGSTKGKKHEQTQVAATAAYRFGNVTPRVSYAHGFKAKVNGVKDANYQYDQVIVGADYDFSKRTSALVSAGWLKQGKGAGKVEQTASMVGLRHKF
XXXXXXXXXXXXXXXXXXIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOmmmMMMMMIIIIIIIIIIIIIMMMMMMMMMMOOOOOOOOOMMMMMMIIIIMMMMMMMIIIIIMMMMMMMMmmmmmmmmmmmmmmmOOOOOOOOOOOOOOOOOOOOOOOMMMMIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMMMI
>BP00310
MYRKSTLAMLIALLTSAASAHAQTDISTIEARLNALEKRLQEAENRAQTAENRAGAAEKKVQQLTAQQQKNQNSTQEVAQRTARLEKKADDKSGFEFHGYARSGVIMNDSGASTKSGAYITPAGETGGAIGRLGNQADTYVEMNLEHKQTLDNGATTRFKVMVADGQTSYNDWTASTSDLNVRQAFVELGNLPTFAGPFKGSTLWAGKRFDRD
NFDIHWIDSDVVFLAGTGGGIYDVKWNDGLRSNFSLYGRNFGDIDDSSNSVQNYILTMNHFAGPLQMMVSGLRAKDNDERKDSNGNLAKGDAANTGVHALLGLHNDSFYGLRDGSSKTALLYGHGLGAEVKGIGSDGALRPGADTWRIASYGTTPLSENWSVAPAMLAQRSKDRYADGDSYQWATFNLRLIQAINQNFALAYEGSYQYMDLKP
EGYNDRQAVNGSFYKLTFAPTFKVGSIGDFFSRPEIRFYTSWMDWSKKLNNYASDDALGSDGFNSGGEWSFGVQMETWF
XXXXXXXXXXXXXXXXXXXXXXIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOOMMMMMMMMIIIIIIIIIIIIIIIIMMMMMMMmmm
mmmmmmmmmmmmmmmMMMMMIIIIIIIIIIIMMMOOOOOOOOOOOOOOOOOOOOMMMMIIIIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIIIIIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOOOOMMMMMIIIIIIIIIIMMMMMMOOOOOOOO
OOOOOOOOOOOOOOOMMMMMIIIIIIIIIIIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMI
>BP00320
MKTRIVSSVTTTLLLGSILMNPVAGAADSDINIKTGTTDIGSNTTVKTGDLVTYDKENGMHKKVFYSFIDDKNHNKKLLVIRTKGTIAGQYRVYSEEGANKSGLAWPSAFKVQLQLPDNEVAQISDYYPRNSIDTKEYMSTLTYGFNGNVTGDDTGKIGGLIGANVSIGHTLKYVQPDFKTILESPTDKKVGWKVIFNNMVNQNWGPYDRDSW
NPVYGNQLFMKTRNGSMKAADNFLDPNKASSLLSSGFSPDFATVITMDRKASKQQTNIDVIYERVRDDYQLHWTSTNWKGTNTKDKWTDRSSERYKIDWEKEEMTN
XXXXXXXXXXXXXXXXXXXXXXXXXXXOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMMMMIIIIMMMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOmI
>BP00339
EISLNGYGRFGLQYVEDRGVGLEDTIISSRLRINIVGTTETDQGVTFGAKLRMQWDDGDAFAGTAGNAAQFWTSYNGVTVSVGNVDTAFDSVALTYDSEMGYEASSFGDAQSSFFAYNSKYDASGALDNYNGIAVTYSISGVNLYLSYVDPDQTVDSSLVTEEFGIAADWSNDMISLAAAYTTDAGGIVDNDIAFVGAAYKFNDAGTVGLNWY
DNGLSTAGDQVTLYGNYAFGATTVRAYVSDIDRAGADTAYGIGADYQFAEGVKVSGSVQSGFANETVADVGVRFDF
IIIIIMMMMMMOOOOOOOOOOOOOOOOOOMMMMMMMMMIIIIIIIIIIIMMMMMMMOOOOOOOOOOOOMMMMMMMMMIIIIIMMMMMMmmmmmmmmmmmmmmmmmmmmmmmmOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMOOOOOOOOOOOOOOOOOOOOMMMMMIIIIIMMMMMOOOOOOOOOOOOOOOOMMMMMIIIIIIIIMMMMMOO
OOOOOOOOOOOMMMMMIIIIIIIIMMMMOOOOOOOOOOOOOOMMMMMIIIIIIIIMMMMOOOOOOOOOOOOOOMMMMMIII
>BP00345
MMKRNILAVIVPALLVAGTANAAEIYNKDGNKVDLYGKAVGLHYFSKGNGENSYGGNGDMTYARLGFKGETQINSDLTGYGQWEYNFQGNNSEGADAQTGNKTRLAFAGLKYADVGSFDYGRNYGVVYDALGYTDMLPEFGGDTAYSDDFFVGRVGGVATYRNSNFFGLVDGLNFAVQYLGKNERDTARRSNGDGVGGSISYEYEGFGIVGAY
GAADRTNLQEAQPLGNGKKAEQWATGLKYDANNIYLAANYGETRNATPITNKFTNTSGFANKTQDVLLVAQYQFDFGLRPSIAYTKSKAKDVEGIGDVDLVNYFEVGATYYFNKNMSTYVDYIINQIDSDNKLGVGSDDTVAVGIVYQF
XXXXXXXXXXXXXXXXXXXXXXIIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOMMMMMMMMIIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOMMMMMMMMIIIIIIMMMMMMMMmmmmmmmmmmmmmmmmmmmmmmmmmmmmOOOOOOOMMMMMIIIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMIIIIMMMMMMMO
```

Figure 21 Top_bp.txt

Source: Enzim.hu. (2014)

Table 2 Sample of 15 TMBs selected out of 123 TMBs available in TOPDB database

Source: Enzim. hu. (2014)

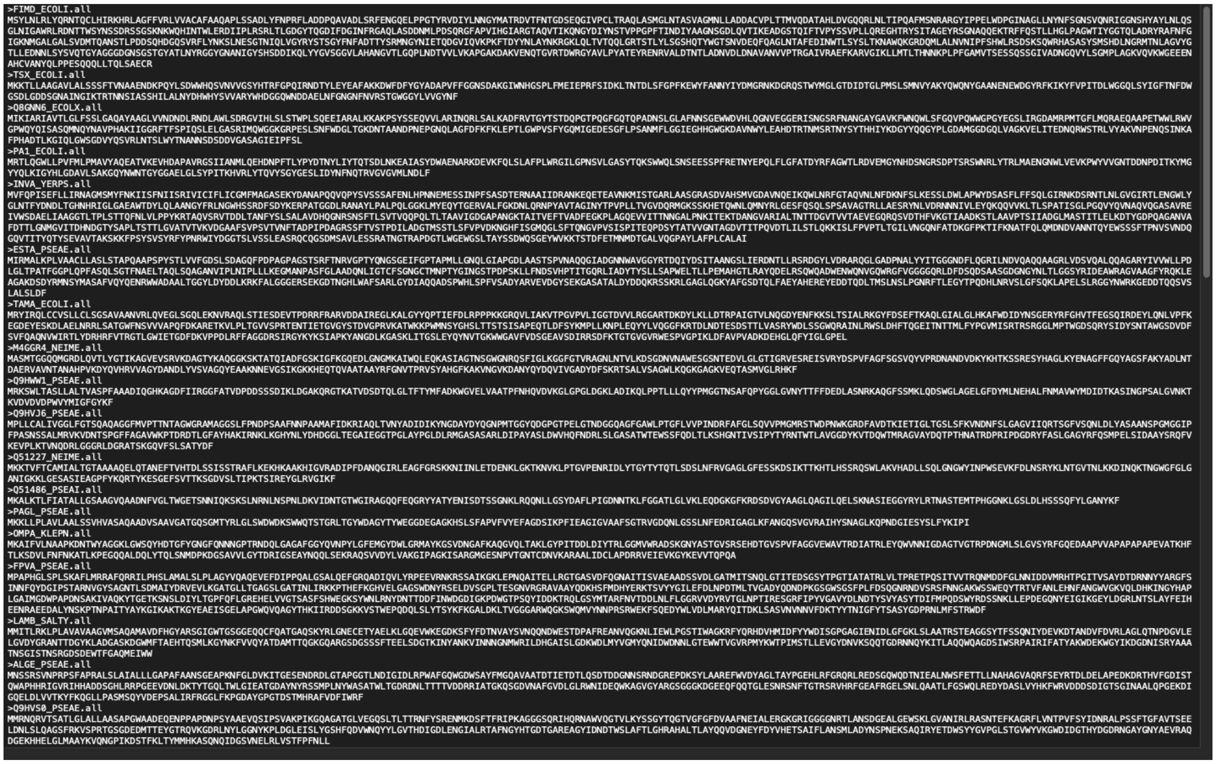


	ID	Description	Organism
1	**BP00056**	Outer membrane usher protein faeD precursor	Escherichia coli
2	**BP00086**	Maltoporin precursor	Escherichia coli
3	**BP00115**	Outer membrane protein A precursor	Escherichia coli
4	**BP00124**	Outer membrane pore protein E precursor	Escherichia coli
5	**BP00193**	Major outer membrane protein P.IA precursor, PIA	Neisseria gonorrhoeae
6	**BP00272**	Major outer membrane protein P.IB precursor, PIB	Neisseria gonorrhoeae
7	**BP00273**	Major outer membrane protein P, PIA	Neisseria meningitidis serogroup B
8	**BP00274**	Outer membrane protein class 2	Neisseria meningitidis
9	**BP00310**	Sucrose porin precursor	Salmonella typhimurium
10	**BP00320**	Alpha-hemolysin precursor (Alpha-toxin) (Alpha-HL)	Staphylococcus aureus
11	**BP00339**	Porin	Rhodobacter blasticus
12	**BP00345**	Outer membrane protein F precursor (Porin ompF) (Outer membrane protein 1A) (Outer membrane protein IA) (Outer membrane protein B)	Escherichia coli
13	**BP00346**	Ferrichrome-iron receptor precursor (Ferric hydroxamate uptake) (Ferric hydroxamate receptor)	Escherichia coli
14	**BP00359**	Outer membrane porin protein 32 precursor (OMP32)	Delftia acidovorans
15	**BP00364**	Outer membrane protein tolC precursor	Escherichia coli

4.4 Data preparation

4.4.1 Data collection

Data were collected from the BOCTOPUS2 dataset. The list of TMB proteins is represented in table 1. It was the dataset used to train the BOCTOPUS2 server as described in the paper of Hayat et al. (Hayat, Peters, Shu, Tsirigos and Elofsson, 2016). The original file boctopus2_dataset_sequenceannotation.txt include all TMB proteins sequences and observed topologies. The sequence is represented in the FASTA format. The FASTA format is commonly used in bioinformatics, and it is a text-based format for representing peptide sequences or nucleotide sequences. A single-letter code corresponds to a single amino acid. M stands for methionine, F for Phenylalanine, for example. The name FASTA originates from the FASTA software package, a DNA and protein sequence alignment software package. There are many different file formats used in bioinformatics. The most common file formats used in sequencing analysis include SAM/BAM format, VCF format, Wig format, BED format or GTF/GFF3 format. Data within boctopus2_dataset_sequenceannotation.txt was divided into two separate files. The first file, boctopus2Sequence.txt, took only the FASTA sequences from boctopus2_dataset_sequenceannotation.txt and adjusted the file and lines numbers using the text editor Sublime Text.



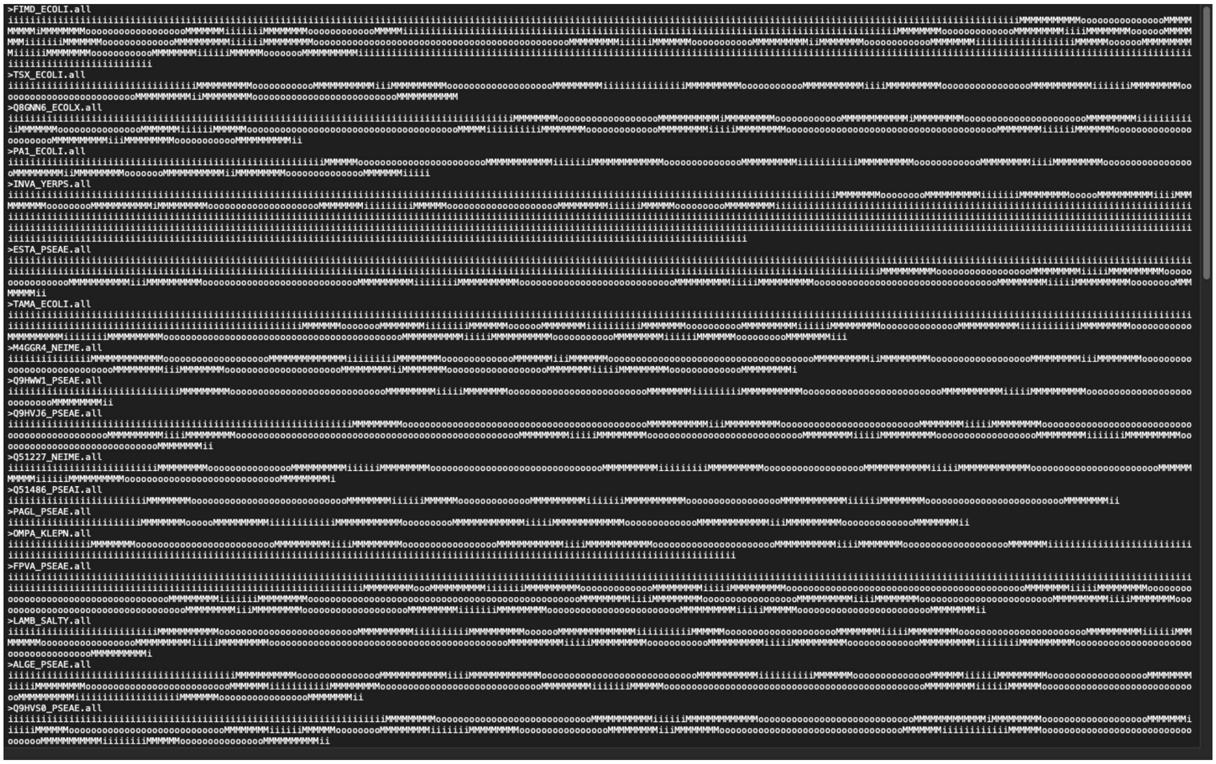
```
>FIMD_ECOLI.all
MSYLNLRLYQRNTQCLHIRKHRLAGFFVRLVVACAFAAQAPLSSADLYFNPRFLADDPQAVADLSRFENGQELPPGTYRVDIYLNNGYMATRDVTFNTGDSEQGIVPCLTRAQLASMGLNTASVAGMNLLADDACVPLTTMVQDATAHLDVGQQRLNLTIPQAFMSNRARGYIPPELWDPGINAGLLNYNFSGNSVQNRIGGNSHYAYLNLQS
GLNIGAWRLRDNTTWSYNSSDRSSGSKNKWQHINTWLERDIIPLRSRLTLGDGYTQGDIFDGINFRGAQLASDDNMLPDSQRGFAPVIHGIARGTAQVTIKQNGYDIYNSTVPPGPFTINDIYAAGNSGDLQVTIKEADGSTQIFTVPYSSVPLLQREGHTRYSITAGEYRSGNAQQEKTRFFQSTLLHGLPAGWTIYGGTQLADRYRAFNFG
IGKNMGALGALSVDMTQANSTLPDDSQHDGQSVRFLYNKSLNESGTNIQLVGYRYSTSGYFNFADTTYSRMNGYNIETQDGVIQVKPKFTDYYNLAYNKRGKLQLTVTQQLGRTSTLYLSGSHQTYWGTSNVDEQFQAGLNTAFEDINWTLSYSLTKNAWQKGRDQMLALNVNIPFSHWLRSDSKSQWRHASASYSMSHDLNGRMTNLAGVYG
TLLEDNNLSYSVQTGYAGGGDGNSGSTGYATLNYRGGYGNANIGYSHSDDIKQLYYGVSGGVLAHANGVTLGQPLNDTVVLVKAPGAKDAKVENQTGVRTDWRGYAVLPYATEYRENRVALDTNTLADNVDLDNAVANVVPTRGAIVRAEFKARVGIKLLMTLTHNNKPLPFGAMVTSESSQSSGIVADNGQVYLSGMPLAGKVQVKWGEEEN
AHCVANYQLPPESQQQLLTQLSAECR
>TSX_ECOLI.all
MKKTLLAAGAVLALSSSFTVNAAENDKPQYLSDWWHQSVNVVGSYHTRFGPQIRNDTYLEYEAFAKKDWFDFYGYADAPVFFGGNSDAKGIWNHGSPLFMEIEPRFSIDKLTNTDLSFGPFKEWYFANNYIYDMGRNKDGRQSTWYMGLGTDIDTGLPMSLSMNVYAKYQWQNYGAANENEWDGYRFKIKYFVPITDLWGGQLSYIGFTNFDW
GSDLGDDSGNAINGIKTRTNNSIASSHILALNYDHWHYSVVARYWHDGGQWNDDAELNFGNGNFNVRSTGWGGYLVVGYNF
>Q8GNN6_ECOLX.all
MIKIARIAVTLGLFSSLGAQAYAAGLVVNDNDLRNDLAWLSDRGVIHLSLSTWPLSQEEIARALKKAKPSYSSEQVVLARINQRLSALKADFRVTGYTSTDQPGTPQGFGQTQPADNSLGLAFNNSGEWWDVHLQGNVEGGERISNGSRFNANGAYGAVKFWNQWLSFGQVPQWWGPGYEGSLIRGDAMRPMTGFLMQRAEQAAPETWWLRWV
GPWQYQISASQMNQYNAVPHAKIIGGRFTFSPIQSLELGASRIMQWGGKGRPESLSNFWDGLTGKDNTAANDPNEPGNQLAGFDFKFKLEPTLGWPVSFYGQMIGEDESGFLPSANMFLGGIEGHHGWGKDAVNWYLEAHDTRTNMSRTNYSYTHHIYKDGYYQQGYPLGDAMGGDGQLVAGKVELITEDNQRWSTRLVYAKVNPENQSINKA
FPHADTLKGIQLGWSGDVYQSVRLNTSLWYTNANNSDSDDVGASAGIEIPFSL
>PA1_ECOLI.all
MRTLQGWLLPVFMLPMAVYAQEATVKEVHDAPAVRGSIIANMLQEHDNPFTLYPYDTNYLIYTQTSDLNKEAIASYDWAENARKDEVKFQLSLAFPLWRGILGPNSVLGASYTQKSWWQLSNSEESSPFRETNYEPQLFLGFATDYRFAGWTLRDVEMGYNHDSNGRSDPTSRSWNRLYTRLMAENGNWLVEVKPWYVVGNTDDNPDITKYMG
YYQLKIGYHLGDAVLSAKGQYNWNTGYGGAELGLSYPITKHVRLYTQVYSGYGESLIDYNFNQTRVGVGVMLNDLF
>INVA_YERPS.all
MVFQPISEFLLIRNAGMSMYFNKIISFNIISRIVICIFLICGMFMAGASEKYDANAPQQVQPYSVSSSAFENLHPNNEMESSINPFSASDTERNAAIIDRANKEQETEAVNKMISTGARLAASGRASDVAHSMVGDAVNQEIKQWLNRFGTAQVNLNFDKNFSLKESSLDWLAPWYDSASFLFFSQLGIRNKDSRNTLNLGVGIRTLENGWLY
GLNTFYDNDLTGHNHRIGLGAEAWTDYLQLAANGYFRLNGWHSSRDFSDYKERPATGGDLRANAYLPALPQLGGKLMYEQYTGERVALFGKDNLQRNPYAVTAGINYTPVPLLTVGVDQRMGKSSKHETQWNLQMNYRLGESFQSQLSPSAVAGTRLLAESRYNLVDRNNNIVLEYQKQQVVKLTLSPATISGLPGQVYQVNAQVQGASAVRE
IVWSDAELIAAGGTLTPLSTTQFNLVLPPYKRTAQVSRVTDDLTANFYSLSALAVDHQGNRSNSFTLSVTVQQPQLTLTAAVIGDGAPANGKTAITVEFTVADFEGKPLAGQEVVITTNNGALPNKITEKTDANGVARIALTNTTDGVTVVTAEVEGQRQSVDTHFVKGTIAADKSTLAAVPTSIIADGLMASTITLELKDTYGDPQAGANVA
FDTTLGNMGVITDHNDGTYSAPLTSTTLGVATVTVKVDGAAFSVPSVTVNFTADPIPDAGRSSFTVSTPDILADGTMSSTLSFVPVDKNGHFISGMQGLSFTQNGVPVSISPITEQPDSYTATVVGNTAGDVTITPQVDTLILSTLQKKISLFPVPTLTGILVNGQNFATDKGFPKTIFKNATFQLQMDNDVANNTQYEWSSSFTPNVSVNDQ
GQVTITYQTYSEVAVTAKSKKFPSYSVSYRFYPNRWIYDGGTSLVSSLEASRQCQGSDMSAVLESSRATNGTRAPDGTLWGEWGSLTAYSSDWQSGEYWVKKTSTDFETMNMDTGALVQGPAYLAFPLCALAI
>ESTA_PSEAE.all
MIRMALKPLVAACLLASLSTAPQAAPSPYSTLVVFGDSLSDAGQFPDPAGPAGSTSRFTNRVGPTYQNGSGEIFGPTAPMLLGNQLGIAPGDLAASTSPVNAQQGIADGNNWAVGGYRTDQIYDSITAANGSLIERDNTLLRSRDGYLVDRARQGLGADPNALYYITGGGNDFLQGRILNDVQAQQAAGRLVDSVQALQQAGARYIVVWLLPD
LGLTPATFGGPLQPFASQLSGTFNAELTAQLSQAGANVIPLNIPLLLKEGMANPASFGLAADQNLIGTCFSGNGCTMNPTYGINGSTPDPSKLLFNDSVHPTITGQRLIADYTYSLLSAPWELTLLPEMAHGTLRAYQDELRSQWQADWENWQNVGQWRGFVGGGGQRLDFDSQDSAASGDGNGYNLTLGGSYRIDEAWRAGVAAGFYRQKLE
AGAKDSDYRMNSYMASAFVQYQENRWWADAALTGGYLDYDDLKRKFALGGGERSEKGDTNGHLWAFSARLGYDIAQQADSPWHLSPFVSADYARVEVDGYSEKGASATALDYDDQKRSSKRLGAGLQGKYAFGSDTQLFAEYAHEREYEDDTQDLTMSLNSLPGNRFTLEGYTPQDHLNRVSLGFSQKLAPELSLRGGYNWRKGEDDTQQSVS
LALSLDF
>TAMA_ECOLI.all
MRYIRQLCCVSLLCLSGSAVAANVRLQVEGLSGQLEKNVRAQLSTIESDEVTPDRRFRARVDDAIREGLKALGYYQPTIEFDLRPPPKKGRQVLIAKVTPGVPVLIGGTDVVLRGGARTDKDYLKLLDTRPAIGTVLNQGDYENFKKSLTSIALRKGYFDSEFTKAQLGIALGLHKAFWDIDYNSGERYRFGHVTFEGSQIRDEYLQNLVPFK
EGDEYESKDLAELNRRLSATGWFNSVVVAPQFDKARETKVLPLTGVVSPRTENTIETGVGYSTDVGPRVKATWKKPWMNSYGHSLTTSTSISAPEQTLDFSYKMPLLKNPLEQYYLVQGGFKRTDLNDTESDSTTLVASRYWDLSSGWQRAINLRWSLDHFTQGEITNTTMLFYPGVMISRTRSRGGLMPTWGDSQRYSIDYSNTAWGSDVDF
SVFQAQNVWIRTLYDRHRFVTRGTLGWIETGDFDKVPPDLRFFAGGDRSIRGYKYKSIAPKYANGDLKGASKLITGSLEYQYNVTGKWWGAVFVDSGEAVSDIRRSDFKTGTGVGVRWESPVGPIKLDFAVPVADKDEHGLQFYIGLGPEL
>M4GGR4_NEIME.all
MASMTGGQQMGRDLQVTLYGTIKAGVEVSRVKDAGTYKAQGGKSKTATQIADFGSKIGFKGQEDLGNGMKAIWQLEQKASIAGTNSGWGNRQSFIGLKGGFGTVRAGNLNTVLKDSGDNVNAWESGSNTEDVLGLGTIGRVESREISVRYDSPVFAGFSGSVQYVPRDNANDVDKYKHTKSSRESYHAGLKYENAGFFGQYAGSFAKYADLNT
DAERVAVNTANAHPVKDYQVHRVVAGYDANDLYVSVAGQYEAAKNNEVGSIKGKKHEQTQVAATAAYRFGNVTPRVSYAHGFKAKVNGVKDANYQYDQVIVGADYDFSKRTSALVSAGWLKQGKGAGKVEQTASMVGLRHKF
>Q9HWW1_PSEAE.all
MRKSWLTASLLLALTVASPFAAADIQGHKAGDFIIRGGFATVDPDDSSSDIKLDGAKQRGTKATVDSDTQLGLTFTYMFADKWGVELVAATPFNHQVDVKGLGPGLDGKLADIKQLPPTLLLQYYPMGGTNSAFQPYGGLGVNYTTFFDEDLASNRKAQGFSSMKLQDSWGLAGELGFDYMLNEHALFNMAVWYMDIDTKASINGPSALGVNKT
KVDVDVDPWVYMIGFGYKF
>Q9HVJ6_PSEAE.all
MPLLCALIVGGLFGTSQAQAGGFMVPTTNTAGWGRAMAGGSLFPNDPSAAFNNPAAMAFIDKRIAQLTVNYADIDIKYNGDAYDYQGNPMTGGYQDGPGTPELGTNDGGQAGFGAWLPTGFLVVPINDRFAFGLSQVVPMGMRSTWDPNWKGRDFAVDTKIETIGLTGSLSFKVNDNFSLGAGVIIQRTSGFVSQNLDLYASAANSPGMGGIP
FPASNSSALMRVKVDNTSPGFFAGAVWKPTDRDTLGFAYHAKIRNKLKGHYNLYDHDGGLTEGAIEGGTPGLAYPGLDLRMGASASARLDIPAYASLDWVHQFNDRLSLGASATWTEWSSFQDLTLKSHGNTIVSIPYTYRNTWTLAVGGDYKVTDQWTMRAGVAYDQTPTHNATRDPRIPDGDRYFASLGAGYRFQSMPELSIDAAYSRQFV
KEVPLKTVNQDRLGGGRLDGRATSKGQVFSLSATYDF
>Q51227_NEIME.all
MKKTVFTCAMIALTGTAAAAQELQTANEFTVHTDLSSISSTRAFLKEKHKAAKHIGVRADIPFDANQGIRLEAGFGRSKKNIINLETDENKLGKTKNVKLPTGVPENRIDLYTGYTYTQTLSDSLNFRVGAGLGFESSKDSIKTTKHTLHSSRQSWLAKVHADLLSQLGNGWYINPWSEVKFDLNSRYKLNTGVTNLKKDINQKTNGWGFGLG
ANIGKKLGESASIEAGPFYKQRTYKESGEFSVTTKSGDVSLTIPKTSIREYGLRVGIKF
>Q51486_PSEAI.all
MKALKTLFIATALLGSAAGVQAADNFVGLTWGETSNNIQKSKSLNRNLNSPNLDKVIDNTGTWGIRAGQQFEQGRYYATYENISDTSSGNKLRQQNLLGSYDAFLPIGDNNTKLFGGATLGLVKLEQDGKGFKRDSDVGYAAGLQAGILQELSKNASIEGGYRYLRTNASTEMTPHGGNKLGSLDLHSSSQFYLGANYKF
>PAGL_PSEAE.all
MKKLLPLAVLAALSSVHVASAQAADVSAAVGATGQSGMTYRLGLSWDWDKSWWQTSTGRLTGYWDAGYTYWEGGDEGAGKHSLSFAPVFVYEFAGDSIKPFIEAGIGVAAFSGTRVGDQNLGSSLNFEDRIGAGLKFANGQSVGVRAIHYSNAGLKQPNDGIESYSLFYKIPI
>OMPA_KLEPN.all
MKAIFVLNAAPKDNTWYAGGKLGWSQYHDTGFYGNGFQNNNGPTRNDQLGAGAFGGYQVNPYLGFEMGYDWLGRMAYKGSVDNGAFKAQGVQLTAKLGYPITDDLDIYTRLGGMVWRADSKGNYASTGVSRSEHDTGVSPVFAGGVEWAVTRDIATRLEYQWVNNIGDAGTVGTRPDNGMLSLGVSYRFGQEDAAPVVAPAPAPAPEVATKHF
TLKSDVLFNFNKATLKPEGQQALDQLYTQLSNMDPKDGSAVVLGYTDRIGSEAYNQQLSEKRAQSVVDYLVAKGIPAGKISARGMGESNPVTGNTCDNVKARAALIDCLAPDRRVEIEVKGYKEVVTQPQA
>FPVA_PSEAE.all
MPAPHGLSPLSKAFLMRRAFQRRILPHSLAMALSLPLAGYVQAQEVEFDIPPQALGSALQEFGRQADIQVLYRPEEVRNKRSSAIKGKLEPNQAITELLRGTGASVDFQGNAITISVAEAADSSVDLGATMITSNQLGTITEDSGSYTPGTIATATRLVLTPRETPQSITVVTRQNMDDFGLNNIDDVMRHTPGITVSAYDTDRNNYYARGFS
INNFQYDGIPSTARNVGYSAGNTLSDMAIYDRVEVLKGATGLLTGAGSLGATINLIRKKPTHEFKGHVELGAGSWDNYRSELDVSGPLTESGNVRGRAVAAYQDKHSFMDHYERKTSVYYGILEFDLNPDTMLTVGADYQDNDPKGSGWSGSFPLFDSQGNRNDVSRSFNNGAKWSSWEQYTRTVFANLEHNFANGWVGKVQLDHKINGYHAP
LGAIMGDWPAPDNSAKIVAQKYTGETKSNSLDIYLTGPFQFLGREHELVVGTSASFSHWEGKSYWNLRNYDNTTDDFINWDGDIGKPDWGTPSQYIDDKTRQLGSYMTARFNVTDDLNLFLGGRVVDYRVTGLNPTIRESGRFIPYVGAVYDLNDTYSVYASYTDIFMPQDSWYRDSSNKLLEPDEGQNYEIGIKGEYLDGRLNTSLAYFEIH
EENRAEEDALYNSKPTNPAITYAYKGIKAKTKGYEAEISGELAPGWQVQAGYTHKIIRDDSGKKVSTWEPQDQLSLYTSYKFKGALDKLTVGGGARWQGKSWQMVYNNPRSRWEKFSQEDYWLVDLMARYQITDKLSASVNVNNVFDKTYYTNIGFYTSASYGDPRNLMFSTRWDF
>LAMB_SALTY.all
MMITLRKLPLAVAVAAGVMSAQAMAVDFHGYARSGIGWTGSGGEQQCFQATGAQSKYRLGNECETYAELKLGQEVWKEGDKSFYFDTNVAYSVNQQNDWESTDPAFREANVQGKNLIEWLPGSTIWAGKRFYQRHDVHMIDFYYWDISGPGAGIENIDLGFGKLSLAATRSTEAGGSYTFSSQNIYDEVKDTANDVFDVRLAGLQTNPDGVLE
LGVDYGRANTTDGYKLADGASKDGWMFTAEHTQSMLKGYNKFVVQYATDAMTTQGKGQARGSDGSSSFTEELSDGTKINYANKVINNNGNMWRILDHGAISLGDKWDLMYVGMYQNIDWDNNLGTEWWTVGVRPMYKWTPIMSTLLEVGYDNVKSQQTGDRNNQYKITLAQQWQAGDSIWSRPAIRIFATYAKWDEKWGYIKDGDNISRYAAA
TNSGISTNSRGDSDEWTFGAQMEIWW
>ALGE_PSEAE.all
MNSSRSVNPRPSFAPRALSLAIALLLGAPAFAANSGEAPKNFGLDVKITGESENDRDLGTAPGGTLNDIGIDLRPWAFGQWGDWSAYFMGQAVAATDTIETDTLQSDTDDGNNSRNDGREPDKSYLAAREFWVDYAGLTAYPGEHLRFGRQRLREDSGQWQDTNIEALNWSFETTLLNAHAGVAQRFSEYRTDLDELAPEDKDRTHVFGDIST
QWAPHHRIGVRIHHADDSGHLRRPGEEVDNLDKTYTGQLTWLGIEATGDAYNYRSSMPLNYWASATWLTGDRDNLTTTTVDDRRIATGKQSGDVNAFGVDLGLRWNIDEQWKAGVGYARGSGGGKDGEEQFQQTGLESNRSNFTGTRSRVHRFGEAFRGELSNLQAATLFGSWQLREDYDASLVYHKFWRVDDDSDIGTSGINAALQPGEKDI
GQELDLVVTKYFKQGLLPASMSQYVDEPSALIRFRGGLFKPGDAYGPGTDSTMHRAFVDFIWRF
>Q9HVS0_PSEAE.all
MMRNQRVTSATLGLALLAASAPGWAADEQENPPAPDNPSYAAEVQSIPSVAKPIKGQAGATGLVEGQSLTLTTRNFYSRENMKDSFTFRIPKAGGGSQRIHQRNAWVQGTVLKYSSGYTQGTVGFGFDVAAFNEIALERGKGRIGGGGNRTLANSDGEALGEWSKLGVANIRLRASNTEFKAGRFLVNTPVFSYIDNRALPSSFTGFAVTSEE
LDNLSLQAGSFRKVSPRTGSGDEDMTTEYGTRQVKGDRLNYLGGNYKPLDGLEISLYGSHFQDVWNQYYLGVTHDIGDLENGIALRTAFNGYHTGDTGAREAGYIDNDTWSLAFTLGHRAHALTLAYQQVDGNEYFDYVHETSAIFLANSMLADYNSPNEKSAQIRYETDWSYYGVPGLSTGVWYVKGWDIDGTHYDGDRNGAYGNYAEVRAQ
DGEKHHELGLMAAYKVQNGPIKDSTFKLTYMMHKASQNQIDGSVNELRLVSTFPFNLL
```

Figure 22 boctopus2Sequence.txt

Source: Enzim.hu. (2014)

boctopus2Labels.txt is the second file used as input for the deep feedforward neural network. This file was created from the observed topologies data in the file boctopus2_dataset_sequenceannotation.txt.

Pore-facing (p) and lipid-facing(l) labels were manually replaced with M to have an i, o, M profile labels for each sequence.



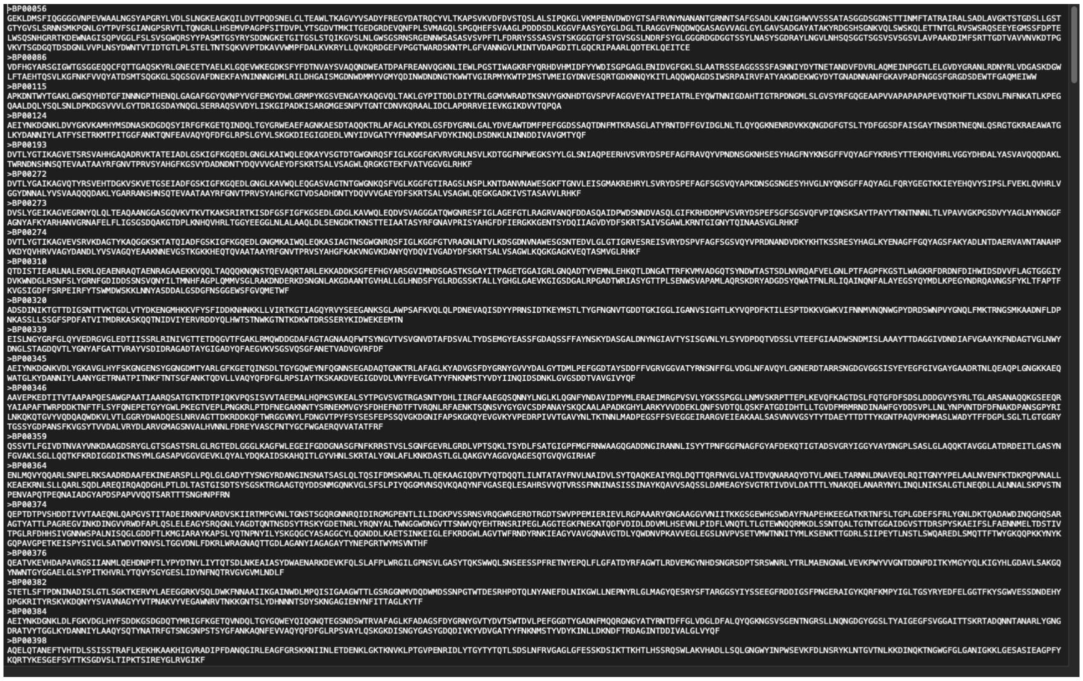
```
>FIMD_ECOLI.all
>TSX_ECOLI.all
>Q8GNN6_ECOLX.all
>PA1_ECOLI.all
>INVA_YERPS.all
>ESTA_PSEAE.all
>TAMA_ECOLI.all
>M4GGR4_NEIME.all
>Q9HWW1_PSEAE.all
>Q9HVJ6_PSEAE.all
>Q51227_NEIME.all
>Q51486_PSEAI.all
>PAGL_PSEAE.all
>OMPA_KLEPN.all
>FPVA_PSEAE.all
>LAMB_SALTY.all
>ALGE_PSEAE.all
>Q9HVS0_PSEAE.all
```

Figure 23 boctopus2Labels.txt

Source: Enzim.hu. (2014)

For the topbp dataset, the file available on the server was also divided into two files.

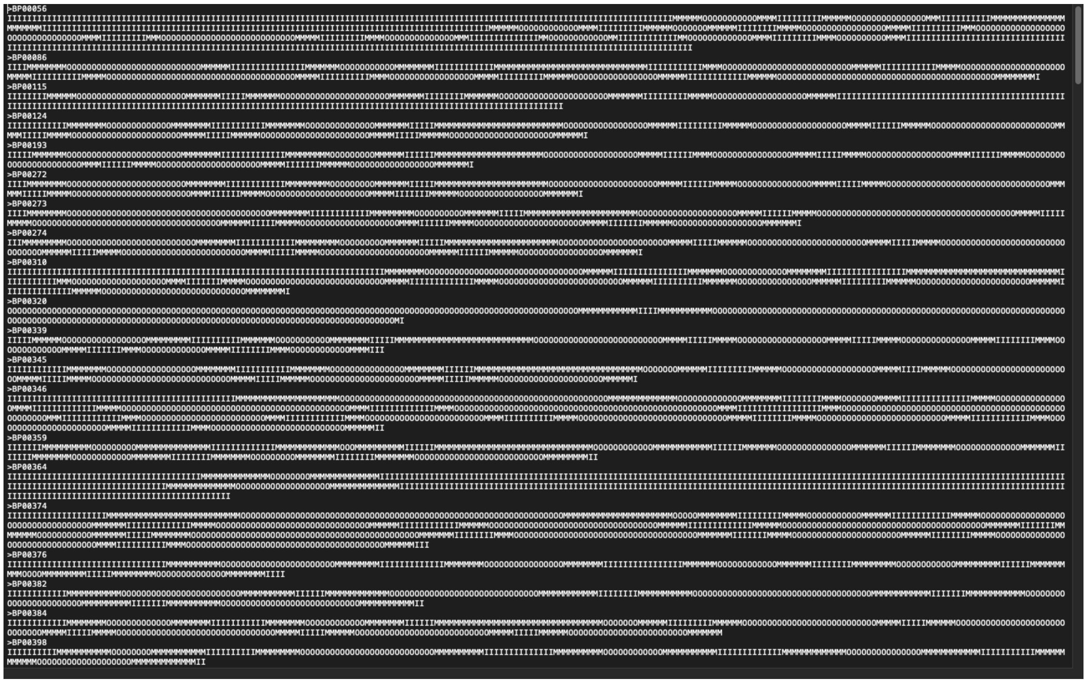
The data file was manually curated. The observed topology represented with an X corresponds to the signal peptide. For the implementations, the signal peptide was ignored. The process of curation was like the BOCTOPUS2 dataset. The topdb_bp.txt was divided into two separate files TOPBPLabels.txt and TOPBPSequence.txt.



```
>BP00056
GEKLDMSFIQGGGGVNPEVWAALNGSYAPGRYLVDLSLNGKEAGKQILDVTPQDSNELCLTEAWLTKAGVYVVSADYFREGYDATRQCYVLTKAPSVKVDFDVSTQSLALSIPQKGLVKMPENVDWDYGTSAFRVNYNANANTGRNNTSAFGSADLKANIGHWVVSSSATASGGDSGDNSTTINMFTATRAIRALSADLAVGKTSTGDSLLGST
GTYGVSLSRNNSMKPGNLGYTPVFSGIANGPSRVTLTQNGRLLHSEMVPAGPFSITDVPLYTSGDVTMKITGEDGRDEVQNFPLSVMAGQLSPGQHEFSVAAGLPDDDSDLKGGVFAASYGYGLDGLTLRAGGVFNQDWQGASAGVVAGLGYLGAVSADGAYATAKYRDGSHSGNKVQLSWSKQLETTNTGLRVSWSRQSEEYEGMSSFDPTE
LWSQSNHGRRTKDEWNAGISQPVGGLFSLSVSGWQRSYYPASMTGSYRYSDDNGKETGITGSLSTQIKGVSLNLGWSGSRNSRGENNWSASASVSVPFTLFDRRYSSSASVSTSKGGGTGFSTGVSGSLNDRFSYGLGGGRDGDGGTSSYLNASYSGDRAYLNGVLNHSQSGGTSGSVSVSGSVLAVPAAKDIMFSRTTGDTVAVVNVKDTPG
VKVTSGDGQTDSDGNLVVPLNSYDWNTVTIDTGTLPLSTELTNTSQKVVPTDKAVVWMPFDALKVKRYLLQVKQRDGEFVPGGTWARDSKNTPLGFVANNGVLMINTVDAPGDITLGQCRIPAARLQDTEKLQEITCE
>BP00086
VDFHGYARSGIGWTGSGGEQQCFQTTGAQSKYRLGNECETYAELKLGQEVWKEGDKSFYFDTNVAYSVAQQNDWEATDPAFREANVQGKNLIEWLPGSTIWAGKRFYQRHDVHMIDFYYWDISGPGAGLENIDVGFGKLSLAATRSSEAGGSSSFASNNIYDYTNETANDVFDVRLAQMEINPGGTLELGVDYGRANLRDNYRLVDGASKDGW
LFTAEHTQSVLKGFNKFVVQYATDSMTSQGKGLSQGSGVAFDNEKFAYNINNNGHMLRILDHGAISMGDNWDMMYVGMYQDINWDNDNGTKWWTVGIRPMYKWTPIMSTVMEIGYDNVESQRTGDKNNQYKITLAQQWQAGDSIWSRPAIRVFATYAKWDEKWGYDYTGNADNNANFGKAVPADFNGGSFGRGDSDEWTFGAQMEIWW
>BP00115
APKDNTWYTGAKLGWSQYHDTGFINNNGPTHENQLGAGAFGGYQVNPYVGFEMGYDWLGRMPYKGSVENGAYKAQGVQLTAKLGYPITDDLDIYTRLGGMVWRADTKSNVYGKNHDTGVSPVFAGGVEYAITPEIATRLEYQWTNNIGDAHTIGTRPDNGMLSLGVSYRFGQGEAAPVVAPAPAPAPEVQTKHFTLKSDVLFNFNKATLKPEG
QAALDQLYSQLSNLDPKDGSVVVLGYTDRIGSDAYNQGLSERRAQSVVDYLISKGIPADKISARGMGESNPVTGNTCDNVKQRAALIDCLAPDRRVEIEVKGIKDVVTQPQA
>BP00124
AEIYNKDGNKLDVYGKVKAMHYMSDNASKDGDQSYIRFGFKGETQINDQLTGYGRWEAEFAGNKAESDTAQQKTRLAFAGLKYKDLGSFDYGRNLGALYDVEAWTDMFPEFGGDSSAQTDNFMTKRASGLATYRNTDFFGVIDGLNLTLQYQGKNENRDVKKQNGDGFGTSLTYDFGGSDFAISGAYTNSDRTNEQNLQSRGTGKRAEAWATG
LKYDANNIYLATFYSETRKMTPITGGFANKTQNFEAVAQYQFDFGLRPSLGYVLSKGKDIEGIGDEDLVNYIDVGATYYFNKNMSAFVDYKINQLDSDNKLNINNDDIVAVGMTYQF
>BP00193
DVTLYGTIKAGVETSRSVAHHGAQADRVKTATEIADLGSKIGFKGQEDLGNGLKAIWQLEQKAYVSGTDTGWGNRQSFIGLKGGFGKVRVGRLNSVLKDTGGFNPWEGKSYYLGLSNIAQPEERHVSVRYDSPEFAGFRAVQYVPNDNSGKNHSESYHAGFNYKNSGFFVQYAGFYKRHSYTTEKHQVHRLVGGYDHDALYASVAVQQQDAKL
TWRNDNSHNSQTEVAATAAYRFGNVTPRVSYAHGFKGSVYDADNDNTYDQVVVGAEYDFSKRTSALVSAGWLQRGKGTEKFVATVGGVGLRHKF
>BP00272
DVTLYGAIKAGVQTYRSVEHTDGKVSKVETGSEIADFGSKIGFKGQEDLGNGLKAVWQLEQGASVAGTNTGWGNKQSFVGLKGGFGTIRAGSLNSPLKNTDANVNAWESGKFTGNVLEISGMAKREHRYLSVRYDSPEFAGFSGSVQYAPKDNSGSNGESYHVGLNYQNSGFFAQYAGLFQRYGEGTKKIEYEHQVYSIPSLFVEKLQVHRLV
GGYDNNALYVSVAAQQQDAKLYGARRANSHNSQTEVAATAAYRFGNVTPRVSYAHGFKGTVDSADHDNTYDQVVVGAEYDFSKRTSALVSAGWLQEGKGADKIVSTASAVVLRHKF
>BP00273
DVSLYGEIKAGVEGRNYQLQLTEAQAANGGASGQVKVTKVTKAKSRIRTKISDFGSFIGFKGSEDLGDGLKAVWQLEQDVSVAGGGATQWGNRESFIGLAGEFGTLRAGRVANQFDDASQAIDPWDSNNDVASQLGIFKRHDDMPVSVRYDSPEFSGFSGSVQFVPIQNSKSAYTPAYYTKNTNNNLTLVPAVVGKPGSDVYYAGLNYKNGGF
AGNYAFKYARHANVGRNAFELFLIGSGSDQAKGTDPLKNHQVHRLTGGYEEGGLNLALAAQLDLSENGDKTKNSTTEIAATASYRFGNAVPRISYAHGFDFIERGKKGENTSYDQIIAGVDYDFSKRTSAIVSGAWLKRNTGIGNYTQINAASVGLRHKF
>BP00274
DVTLYGTIKAGVEVSRVKDAGTYKAQGGKSKTATQIADFGSKIGFKGQEDLGNGMKAIWQLEQKASIAGTNSGWGNRQSFIGLKGGFGTVRAGNLNTVLKDSGDNVNAWESGSNTEDVLGLGTIGRVESREISVRYDSPVFAGFSGSVQYVVPRDNANDVDKYKHTKSSRESYHAGLKYENAGFFGQYAGSFAKYADLNTDAERVAVNTANAHP
VKDYQVHRVVAGYDANDLYVSVAGQYEAAKNNEVGSTKGKKHEQTQVAATAAYRFGNVTPRVSYAHGFKAKVNGVKDANYQYDQVIVGADYDFSKRTSALVSAGWLKQGKGAGKVEQTASMVGLRHKF
>BP00310
QTDISTIEARLNALEKRLQEAENRAQTAENRAGAEEKKVQQLTAQQQKNQNSTQEVAQRTARLEKKADDKSGFEFHGYARSGVIMNDSGASTKSGAYITPAGETGGAIGRLGNQADTYVEMNLEHKQTLDNGATTRFKVMVADGQTSYNDWTASTSDLNVRQAFVELGNLPTFAGPFKGSTLWAGKRFDRDNFDIHWIDSDVVFLAGTGGGIY
DVKWNDGLRSNFSLYGRNFGDIDDSSNSVQNYILTMNHFAGPLQMMVSGLRAKDNDERKDSNGNLAKGDAANTGVHALLGLHNDSFYGLRDGSSKTALLYGHGLGAEVKGIGSDGALRPGADTWRIASYGTTPLSENWSVAPAMLAQRSKDRYADGDSYQWATFNLRLIQAINQNFALAYEGSYQYMDLKPEGYNDRQAVNGSFYKLTFAPTF
KVGSIGDFFSRPEIRFYTSWMDWSKKLNNYASDDALGSDGFNSGGEWSFGVQMETWF
>BP00320
ADSDINIKTGTTDIGSNTTVKTGDLVTYDKENGMHKKVFYSFIDDKNHNKKLLVIRTKGTIAGQYRVYSEEGANKSGLAWPSAFKVQLQLPDNEVAQISDYYPRNSIDTKEYMSTLTYGFNGNVTGDDTGKIGGLIGANVSIGHTLKYVQPDFKTILESPTDKKVGWKVIFNNMVNQNWGPYDRDSWNPVYGNQLFMKTRNGSMKAADNFLDP
NKASSLLSSGFSPDFATVITMDRKASKQQTNIDVIYERVRDDYQLHWTSTNWKGTNTKDKWTDRSSERYKIDWEKEEMTN
>BP00339
EISLNGYGRFGLQYVEDRGVGLEDTIISSRLRINIVGTTETDQGVTFGAKLRMQWDDGDAFAGTAGNAAQFWTSYNGVTVSVGNVDTAFDSVALTYDSEMGYEASSFGDAQSSFFAYNSKYDASGALDNYNGIAVTYSISGVNLYLSYVDPDQTVDSSLVTEEFGIAADWSNDMISLAAAYTTDAGGIVDNDIAFVGAAYKFNDAGTVGLNWY
DNGLSTAGDQVTLYGNYAFGATTVRAYVSDIDRAGADTAYGIGADYQFAEGVKVSGSVQSGFANETVADVGVRFDF
>BP00345
AEIYNKDGNKVDLYGKAVGLHYFSKGNGENSYGGNGDMTYARLGFKGETQINSDLTGYGQWEYNFQGNNSEGADAQTGNKTRLAFAGLKYADVGSFDYGRNYGVVYDALGYTDMLPEFGGDTAYSDDFFVGRVGGVATYRNSNFFGLVDGLNFAVQYLGKNERDTARRSNGDGVGGSISYEYEGFGIVGAYGAADRTNLQEAQPLGNGKKAEQ
WATGLKYDANNIYLAANYGETRNATPITNKFTNTSGFANKTQDVLLVAQYQFDFGLRPSIAYTKSKAKDVEGIGDVDLVNYFEVGATYYFNKNMSTYVDYIINQIDSDNKLGVGSDDTVAVGIVYQF
>BP00346
AAVEPKEDTITVTAAPAPQESAWGPAATIAARQSATGTKTDTPIQKVPQSISVVTAEEMALHQPKSVKEALSYTPGVSVGTRGASNTYDHLIIRGFAAEGQSQNNYLNGLKLQGNFYNDAVIDPYMLERAEIMRGPVSVLYGKSSPGGLLNMVSKRPTTEPLKEVQFKAGTDSLFQTGFDFSDSLDDDGVYSYRLTGLARSANAQQKGSEEQR
YAIAPAFTWRPDDKTNFTFLSYFQNEPETGYYGWLPKEGTVEPLPNGKRLPTDFNEGAKNNTYSRNEKMVGYSFDHEFNDTFTVRQNLRFAENKTSQNSVYGYGVCSDPANAYSKQCAALAPADKGHYLARKYVVDDEKLQNFSVDTQLQSKFATGDIDHTLLTGVDFMRMRNDINAWFGYDDSVPLLNLYNPVNTDFDFNAKDPANSGPYRI
LNKQKQTGVYVVQDQAQWDKVLVTLGGRYDWADQESLNRVAGTTDKRDDKQFTWRGGVNYLFDNGVTPYFSYSESFEPSSQVGKDGNIFAPSKGKQYEVGVKYVPEDRPIVVTGAVYNLTKTNNLMADPEGSFFSVEGGEIRARGVEIEAKAALSASVNVVGSYTYTDAEYTTDTTYKGNTPAQVPKHMASLWADYTFFDGPLSGLTLGTGGRY
TGSSYGDPANSFKVGSYTVVDALVRYDLARVGMAGSNVALHVNNLFDREYVASCFNTYGCFWGAERQVVATATFRF
>BP00359
QSSVTLFGIVDTNVAYVNKDAAGDSRYGLGTSGASTSRLGLRGTEDLGGGLKAGFWLEGEIFGDDGNASGFNFKRRSTVSLSGNFGEVRLGRDLVPTSQKLTSYDLFSATGIGPFMGFRNWAAGQGADDNGIRANNLISYYTPNFGGFNAGFGYAFDEKQTIGTADSVGRYIGGYVAYDNGPLSASLGLAQQKTAVGGLATDRDEITLGASYN
FGVAKLSGLLQQTKFKRDIGGDIKTNSYMLGASAPVGGVGEVKLQYALYDQKAIDSKAHQITLGYVHNLSKRTALYGNLAFLKNKDASTLGLQAKGVYAGGVQAGESQTGVQVGIRHAF
>BP00364
ENLMQVYQQARLSNPELRKSAADRDAAFEKINEARSPLLPQLGLGADYTYSNGYRDANGINSNATSASLQLTQSIFDMSKWRALTLQEKAAGIQDVTYQTDQQTLILNTATAYFNVLNAIDVLSYTQAQKEAIYRQLDQTTQRFNVGLVAITDVQNARAQYDTVLANELTARNNLDNAVEQLRQITGNYYPELAALNVENFKTDKPQPVNALL
KEAEKRNLSLLQARLSQDLAREQIRQAQDGHLPTLDLTASTGISDTSYSGSKTRGAAGTQYDDSNMGQNKVGLSFSLPIYQGGMVNSQVKQAQYNFVGASEQLESAHRSVVQTVRSSFNNINASISSINAYKQAVVSAQSSLDAMEAGYSVGTRTIVDVLDATTTLYNAKQELANARYNYLINQLNIKSALGTLNEQDLLALNNALSKPVSTN
PENVAPQTPEQNAIADGYAPDSPAPVVQQTSARTTTSNGHNPFRN
>BP00374
QEPTDTPVSHDDTIVVTAAEQNLQAPGVSTITADEIRKNPVARDVSKIIRTMPGVNLTGNSTSGQRGNNRQIDIRGMGPENTLILIDGKPVSSRNSVRQGWRGERDTRGDTSWVPPEMIERIEVLRGPAAARYGNGAAGGVVNIITKKGSGEWHGSWDAYFNAPEHKEEGATKRTNFSLTGPLGDEFSFRLYGNLDKTQADAWDINQGHQSAR
AGTYATTLPAGREGVINKDINGVVRWDFAPLQSLELEAGYSRQGNLYAGDTQNTNSDSYTRSKYGDETNRLYRQNYALTWNGGWDNGVTTSNWVQYEHTRNSRIPEGLAGGTEGKFNEKATQDFVDIDLDDVMLHSEVNLPIDFLVNQTLTLGTEWNQQRMKDLSSNTQALTGTNTGGAIDGVSTTDRSPYSKAEIFSLFAENNMELTDSTIV
TPGLRFDHHSIVGNNWSPALNISQGLGDDFTLKMGIARAYKAPSLYQTNPNYILYSKGQGCYASAGGCYLQGNDDLKAETSINKEIGLEFKRDGWLAGVTWFRNDYRNKIEAGYVAVGQNAVGTDLYQWDNVPKAVVEGLEGSLNVPVSETVMWTNNITYMLKSENKTTGDRLSIIPEYTLNSTLSWQAREDLSMQTTFTWYGKQQPKKYNYK
GQPAVGPETKEISPYSIVGLSATWDVTKNVSLTGGVDNLFDKRLWRAGNAQTTGDLAGANYIAGAGAYTYNEPGRTWYMSVNTHF
>BP00376
QEATVKEVHDAPAVRGSIIANMLQEHDNPFTLYPYDTNYLIYTQTSDLNKEAIASYDWAENARKDEVKFQLSLAFPLWRGILGPNSVLGASYTQKSWWQLSNSEESSPFRETNYEPQLFLGFATDYRFAGWTLRDVEMGYNHDSNGRSDPTSRSWNRLYTRLMAENGNWLVEVKPWYVVGNTDDNPDITKYMGYYQLKIGYHLGDAVLSAKGQ
YNWNTGYGGAELGLSYPITKHVRLYTQVYSGYGESLIDYNFNQTRVGVGVMLNDLF
>BP00382
STETLSFTPDNINADISLGTLSGKTKERVYLAEEGGRKVSQLDWKFNNAAIIKGAINWDLMPQISIGAAGWTTLGSRGGNMVDQDWMDSSNPGTWTDESRHPDTQLNYANEFDLNIKGWLLNEPNYRLGLMAGYQESRYSFTARGGSYIYSSEEGFRDDIGSFPNGERAIGYKQRFKMPYIGLTGSYRYEDFELGGTFKYSGWVESSDNDEHY
DPGKRITYRSKVKDQNYYSVAVNAGYYVTPNAKVYVEGAWNRVTNKKGNTSLYDHNNNTSDYSKNGAGIENYNFITTAGLKYTF
>BP00384
AEIYNKDGNKLDLFGKVDGLHYFSDDKGSDGDQTYMRIGFKGETQVNDQLTGYGQWEYQIQGNQTEGSNDSWTRVAFAGLKFADAGSFDYGRNYGVTYDVTSWTDVLPEFGGDTYGADNFMQQRGNGYATYRNTDFFGLVDGLDFALQYQGKNGSVSGENTNGRSLLNQNGDGYGGSLTYAIGEGFSVGGAITTSKRTADQNNTANARLYGNG
DRATVYTGGLKYDANNIYLAAQYSQTYNATRFGTSNGSNPSTSYGFANKAQNFEVVAQYQFDFGLRPSVAYLQSKGKDISNGYGASYGDQDIVKYVDVGATYYFNKNMSTYVDYKINLLDKNDFTRDAGINTDDIVALGLVYQF
>BP00398
AQELQTANEFTVHTDLSSISSTRAFLKEKHKAAKHIGVRADIPFDANQGIRLEAGFGRSKKNIINLETDENKLGKTKNVKLPTGVPENRIDLYTGYTYTQTLSDSLNFRVGAGLGFESSKDSIKTTKHTLHSSRQSWLAKVHADLLSQLGNGWYINPWSEVKFDLNSRYKLNTGVTNLKKDINQKTNGWGFGLGANIGKKLGESASIEAGPFY
KQRTYKESGEFSVTTKSGDVSLTIPKTSIREYGLRVGIKF
```

Figure 24 TOPBPSequence.txt

Source: Enzim.hu. (2014)



```
>BP00056
[illegible]
[illegible]
[illegible]
[illegible]
>BP00086
[illegible]
[illegible]
>BP00115
[illegible]
[illegible]
>BP00124
[illegible]
[illegible]
>BP00193
[illegible]
[illegible]
>BP00272
[illegible]
[illegible]
>BP00273
[illegible]
[illegible]
>BP00274
[illegible]
[illegible]
>BP00310
[illegible]
[illegible]
[illegible]
>BP00320
[illegible]
[illegible]
>BP00339
[illegible]
[illegible]
>BP00345
[illegible]
[illegible]
>BP00346
[illegible]
[illegible]
[illegible]
[illegible]
>BP00359
[illegible]
[illegible]
>BP00364
[illegible]
[illegible]
[illegible]
>BP00374
[illegible]
[illegible]
[illegible]
[illegible]
>BP00376
[illegible]
[illegible]
>BP00382
[illegible]
[illegible]
>BP00384
[illegible]
[illegible]
>BP00398
[illegible]
[illegible]
```

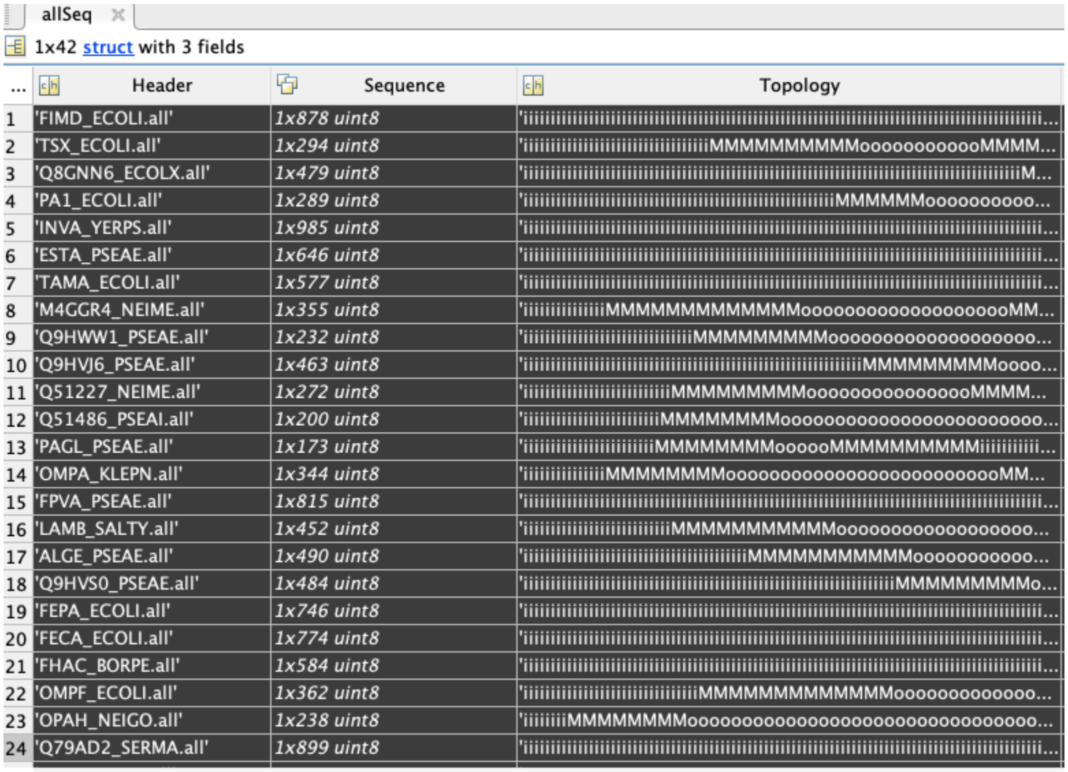
Figure 25 TOPBPLabels.txt

Source: Enzim.hu. (2014)

4.4.2 Data preprocessing

4.4.2.1 Boctopus2 structure array

A dataset needs to be created and formatted for MATLAB to train the deep feedforward neural network. A 1x42 structure array with three fields (header, sequence, topology) has been created for the boctopus2dataset.mat.



1x42 struct with 3 fields

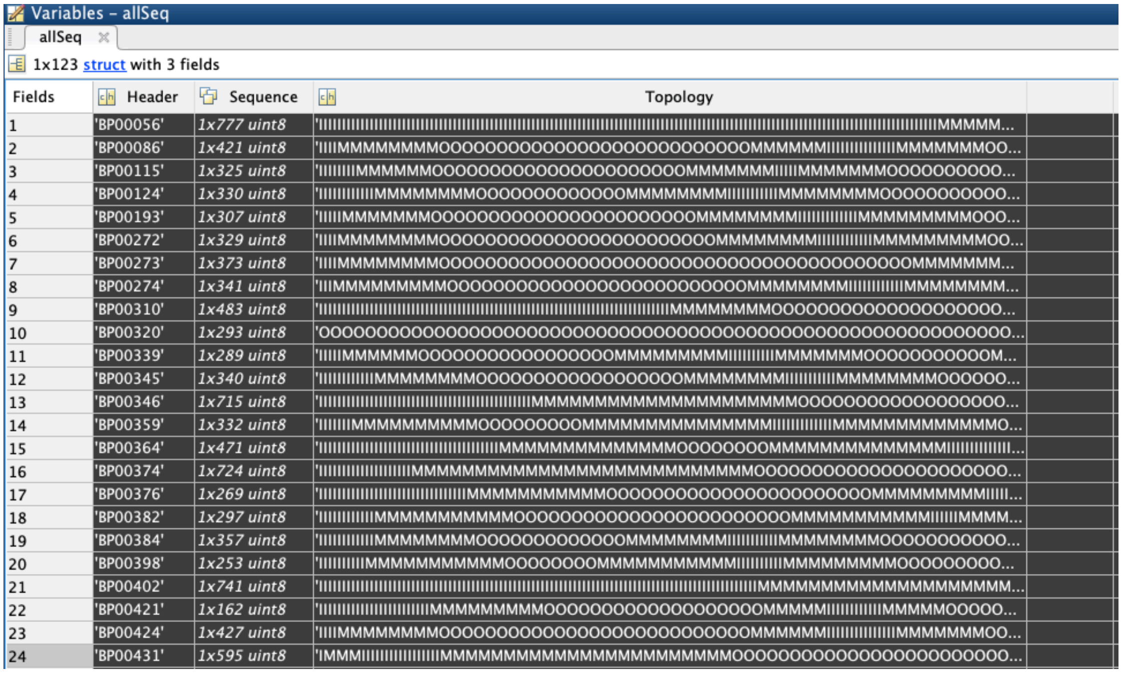
...	Header	Sequence	Topology
1	'FIMD_ECOLI.all'	1x878 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...
2	'TSX_ECOLI.all'	1x294 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiMMMMMMMMMMooooooooooMMMM...
3	'Q8GNN6_ECOLX.all'	1x479 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiM...
4	'PA1_ECOLI.all'	1x289 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiMMMMMMoooooooooo...
5	'INVA_YERPS.all'	1x985 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...
6	'ESTA_PSEAE.all'	1x646 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...
7	'TAMA_ECOLI.all'	1x577 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...
8	'M4GGR4_NEIME.all'	1x355 uint8	'iiiiiiiiiiiiiMMMMMMMMMMMMoooooooooooooooooooMM...
9	'Q9HWW1_PSEAE.all'	1x232 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiMMMMMMMMMoooooooooooooooooo...
10	'Q9HVJ6_PSEAE.all'	1x463 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiMMMMMMMMMoooo...
11	'Q51227_NEIME.all'	1x272 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiMMMMMMMMMooooooooooooooMMMM...
12	'Q51486_PSEAI.all'	1x200 uint8	'iiiiiiiiiiiiiiiiiiiiiiMMMMMMMMoooooooooooooooooooooooo...
13	'PAGL_PSEAE.all'	1x173 uint8	'iiiiiiiiiiiiiiiiiiiiiMMMMMMMMoooooMMMMMMMMMMiiiiiiiii...
14	'OMPA_KLEPN.all'	1x344 uint8	'iiiiiiiiiiiiiMMMMMMMMooooooooooooooooooooooooMM...
15	'FPVA_PSEAE.all'	1x815 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...
16	'LAMB_SALTY.all'	1x452 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiMMMMMMMMMMMoooooooooooooooooo...
17	'ALGE_PSEAE.all'	1x490 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiMMMMMMMMMMMoooooooooooo...
18	'Q9HVS0_PSEAE.all'	1x484 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiMMMMMMMMMo...
19	'FEPA_ECOLI.all'	1x746 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...
20	'FECA_ECOLI.all'	1x774 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...
21	'FHAC_BORPE.all'	1x584 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...
22	'OMPF_ECOLI.all'	1x362 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiMMMMMMMMMMMMMoooooooooooooo...
23	'OPAH_NEIGO.all'	1x238 uint8	'iiiiiiiMMMMMMMMooooooooooooooooooooooooooooooo...
24	'Q79AD2_SERMA.all'	1x899 uint8	'iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii...

Figure 26 boctopus2dataset.m

42 corresponds to the 42 TMB sequences. ‘header’ field corresponds to the annotation of a given protein sequence, ‘sequence’ corresponds to the protein sequence and ‘topology’ corresponds to the predicted topology.

4.4.2.2 Topbp structure array

Similarly, a 1x123structure array with three fields (header, sequence, topology) has been created for the Topdb dataset named topbpdataset.m



Variables – allSeq

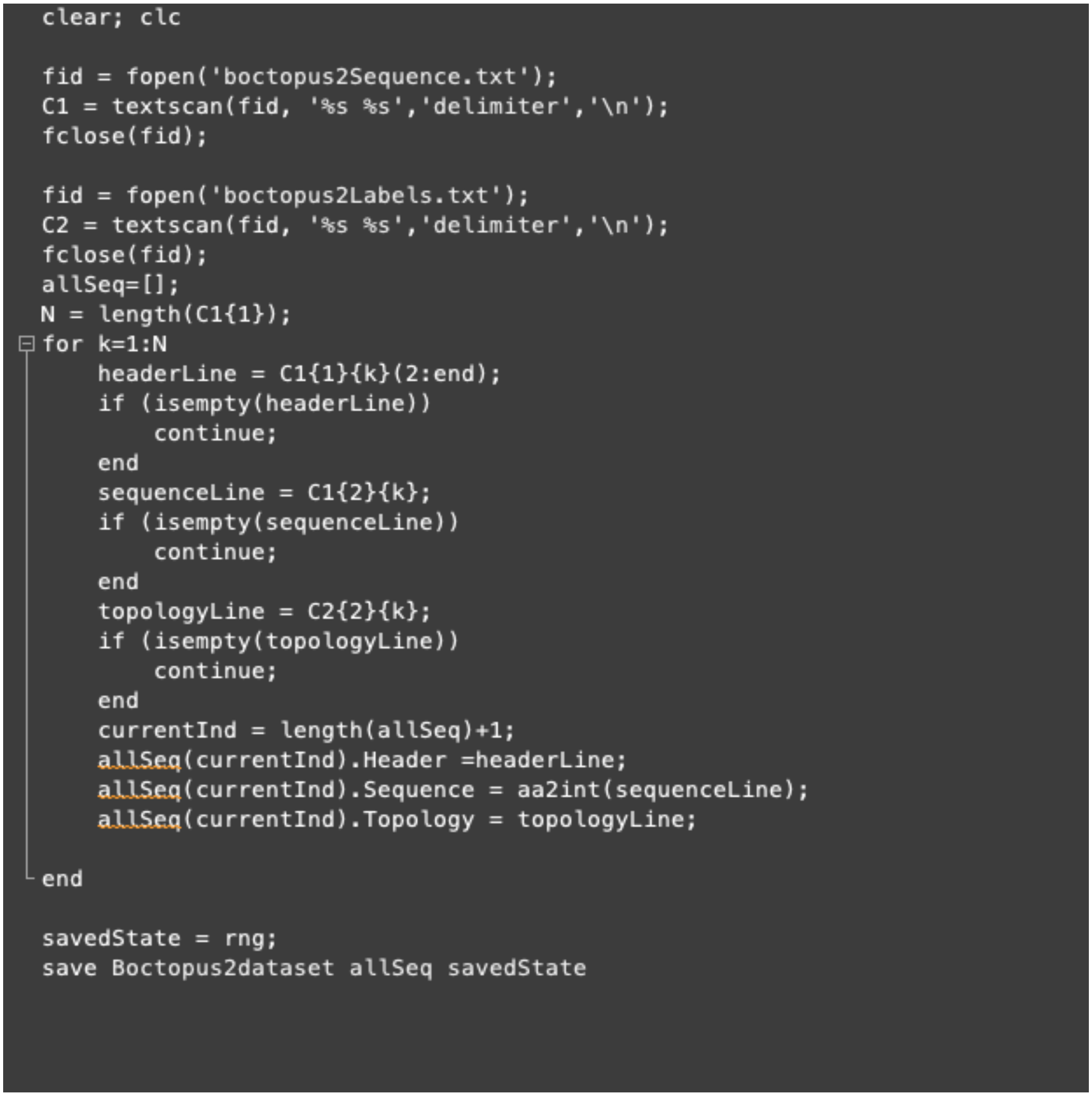
allSeq

1x123 struct with 3 fields

Fields	Header	Sequence	Topology
1	'BP00056'	1x777 uint8	'IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMM...
2	'BP00086'	1x421 uint8	'IIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMIIIIIIIIIIIIIIIIMMMMMMMMOO...
3	'BP00115'	1x325 uint8	'IIIIIIIIMMMMMMOOOOOOOOOOOOOOOOOOOOOOMMMMMMMIIIIIMMMMMMMOOOOOOOOOOO...
4	'BP00124'	1x330 uint8	'IIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOMMMMMMMMIIIIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOO...
5	'BP00193'	1x307 uint8	'IIIIIMMMMMMMOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMIIIIIIIIIIIIIIIIMMMMMMMMMOOO...
6	'BP00272'	1x329 uint8	'IIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMIIIIIIIIIIIIIIMMMMMMMMMOO...
7	'BP00273'	1x373 uint8	'IIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMM...
8	'BP00274'	1x341 uint8	'IIIMMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMIIIIIIIIIIIIIMMMMMMMMM...
9	'BP00310'	1x483 uint8	'IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOOO...
10	'BP00320'	1x293 uint8	'OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO...
11	'BP00339'	1x289 uint8	'IIIIIMMMMMMMOOOOOOOOOOOOOOOOMMMMMMMMMMIIIIIIIIIIMMMMMMMOOOOOOOOOOOOOOM...
12	'BP00345'	1x340 uint8	'IIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOMMMMMMMMMIIIIIIIIIIIMMMMMMMMMOOOOOOO...
13	'BP00346'	1x715 uint8	'IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMMMMMMMMMMMMMMMOOOOOOOOOOOOOOOOOOOO...
14	'BP00359'	1x332 uint8	'IIIIIIIMMMMMMMMMMMOOOOOOOOOMMMMMMMMMMMMMMMMMIIIIIIIIIIIMMMMMMMMMMMMMMMO...
15	'BP00364'	1x471 uint8	'IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMMMMMMMMOOOOOOOOMMMMMMMMMMMMMMMIIIIIIIIIIIIIIIII...
16	'BP00374'	1x724 uint8	'IIIIIIIIIIIIIIIIIIIIMMMMMMMMMMMMMMMMMMMMMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOO...
17	'BP00376'	1x269 uint8	'IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMMMIIIIII...
18	'BP00382'	1x297 uint8	'IIIIIIIIIIIIMMMMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMMMMIIIIIIMMMM...
19	'BP00384'	1x357 uint8	'IIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOMMMMMMMMMIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOO...
20	'BP00398'	1x253 uint8	'IIIIIIIIIIMMMMMMMMMMMMOOOOOOOOMMMMMMMMMMMMIIIIIIIIIIMMMMMMMMMMOOOOOOOOOO...
21	'BP00402'	1x741 uint8	'IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMMMMMMMMMMMMMMM...
22	'BP00421'	1x162 uint8	'IIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOMMMMMMIIIIIIIIIIIMMMMMMOOOOO...
23	'BP00424'	1x427 uint8	'IIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMIIIIIIIIIIIIIIIIMMMMMMMMOO...
24	'BP00431'	1x595 uint8	'IMMMIIIIIIIIIIIIIIIMMMMMMMMMMMMMMMMMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOO...

Figure 27 topbpdataset. mat.

To create the structure array for the datasets, a file named comput was written in MATLAB. It is represented in figure 28.



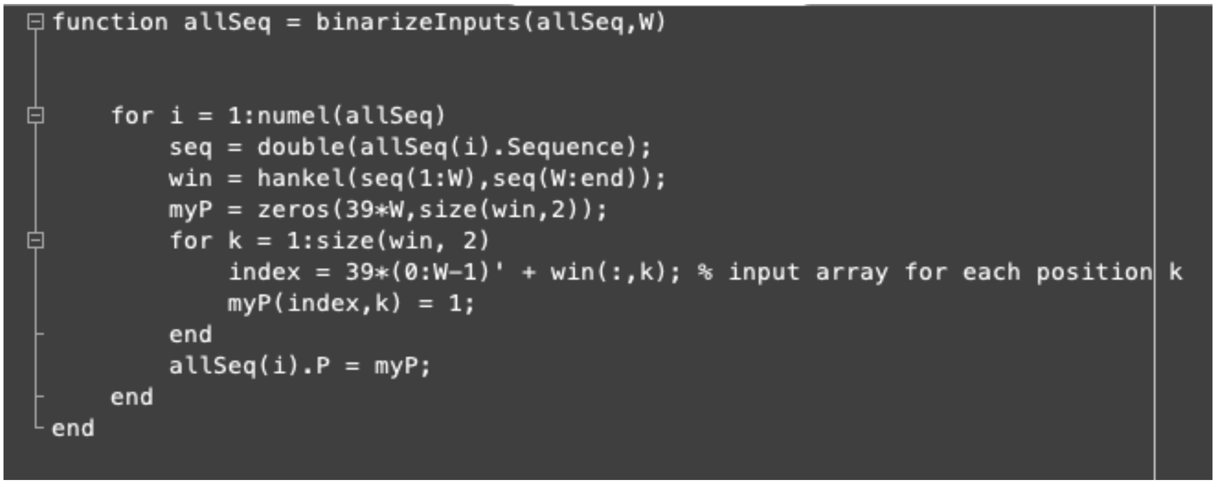
```
clear; clc

fid = fopen('boctopus2Sequence.txt');
C1 = textscan(fid, '%s %s','delimiter','\n');
fclose(fid);

fid = fopen('boctopus2Labels.txt');
C2 = textscan(fid, '%s %s','delimiter','\n');
fclose(fid);
allSeq=[];
N = length(C1{1});
for k=1:N
    headerLine = C1{1}{k}(2:end);
    if (isempty(headerLine))
        continue;
    end
    sequenceLine = C1{2}{k};
    if (isempty(sequenceLine))
        continue;
    end
    topologyLine = C2{2}{k};
    if (isempty(topologyLine))
        continue;
    end
    currentInd = length(allSeq)+1;
    allSeq(currentInd).Header =headerLine;
    allSeq(currentInd).Sequence = aa2int(sequenceLine);
    allSeq(currentInd).Topology = topologyLine;

end

savedState = rng;
save Boctopus2dataset allSeq savedState
```

Figure 28 Comput file

4.4.3 Data transformation

4.4.3.1 Binarisation of the inputs and targets

The function binarizeInputs makes the binarisation of the inputs.



```
function allSeq = binarizeInputs(allSeq,W)


    for i = 1:numel(allSeq)
        seq = double(allSeq(i).Sequence);
        win = hankel(seq(1:W),seq(W:end));
        myP = zeros(39*W,size(win,2));
        for k = 1:size(win, 2)
            index = 39*(0:W-1)' + win(:,k); % input array for each position k
            myP(index,k) = 1;
        end
        allSeq(i).P = myP;
    end
end
```

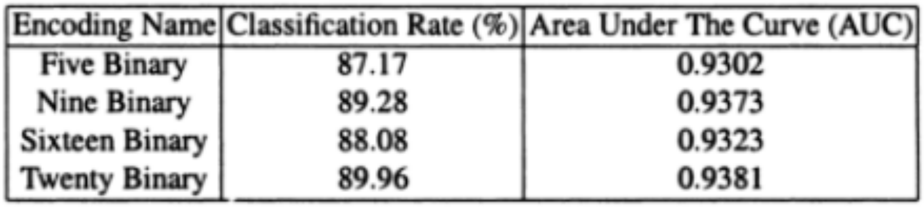
Figure 29 BinarizeInputs.m

In the above code, all the possible subsequences corresponding to a sliding window of size W are determined by creating a Hankel matrix for each protein sequence.

Twenty binary bits are the most common distributed encoding method. Each amino acid is represented by a unique 20-bit binary string that consists of ‘0’ and one ‘1’. Table 3 represents 20-bits encoding for 20 different amino acids. Conversion of the amino acids sequences into real numbers is necessary to obtain numerical input vectors. In a study, ANNs trained with 20-binary-bit encoding produced the highest classification rate and AUC, therefore superior robustness. They observed a minor effect on the classification rate and AUC when either altering the bit length or binary input data (From 5-bits to 20-bits) or increasing the hidden layer nodes after a certain level or both.

Table 3 Best classification rate and area under the ROC curve on different binary encoding

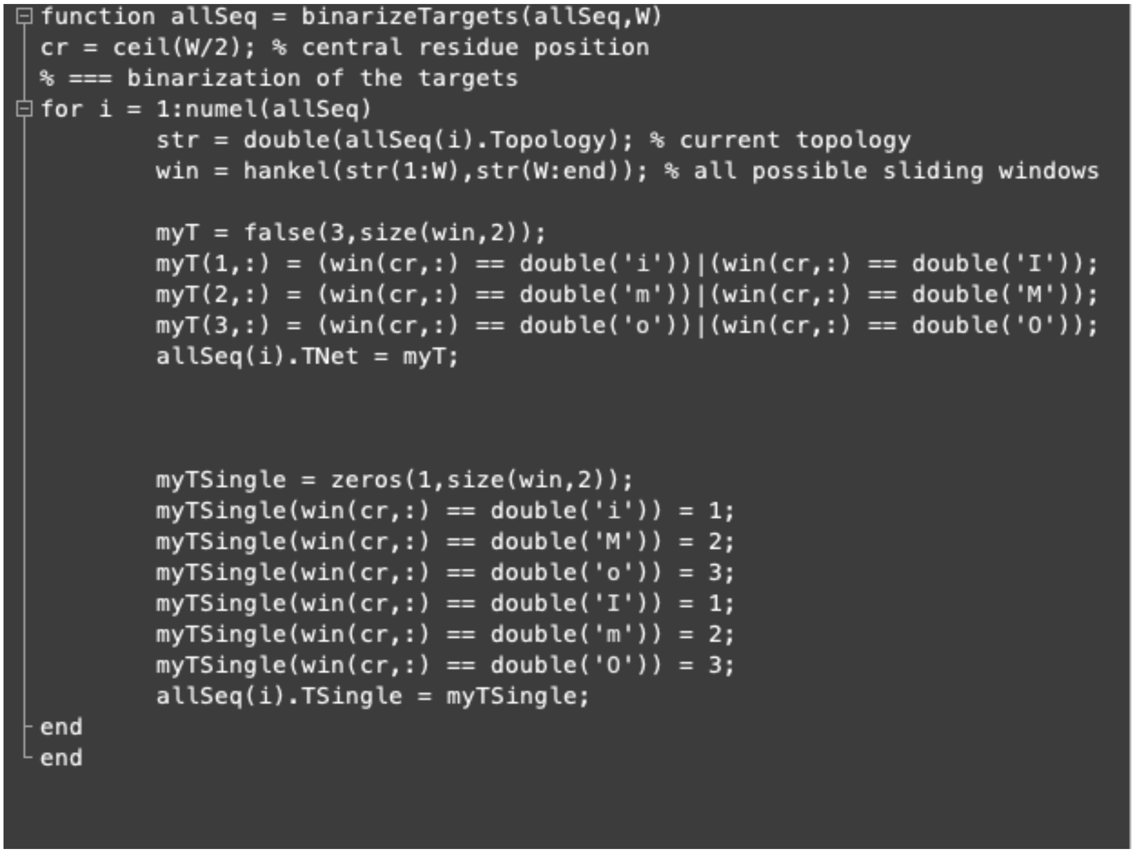
Source: Singh, S., Singh, M. (2007)



Encoding Name	Classification Rate (%)	Area Under The Curve (AUC)
Five Binary	87.17	0.9302
Nine Binary	89.28	0.9373
Sixteen Binary	88.08	0.9323
Twenty Binary	89.96	0.9381

When binary encoding is used, no Physico-chemical properties are taken into consideration. Previous studies also suggest that the addition of Physico-chemical properties does not always increase neural network performances (Brusic, Rudy and Harrison, 1995). Many beta-barrel consists of alternating hydrophobic and hydrophilic side chains. It would not be helpful to apply simple hydrophobic scales for the topology prediction of TMB; therefore, this approach is not considered in this paper. Various bits encoding has been used as part of the computation.

The function binarizeTargets makes the binarisation of the targets and is represented below.



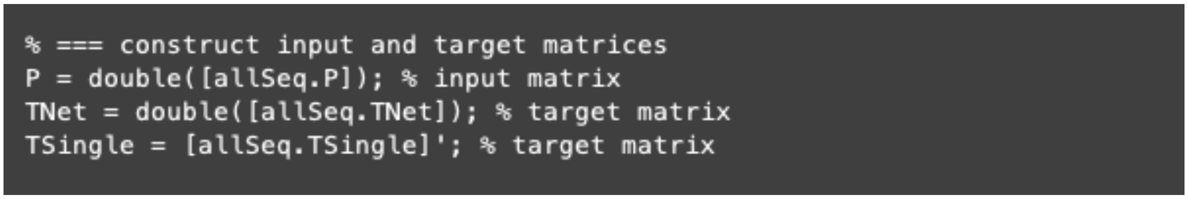
```
function allSeq = binarizeTargets(allSeq,W)
cr = ceil(W/2); % central residue position
% === binarization of the targets
for i = 1:numel(allSeq)
        str = double(allSeq(i).Topology); % current topology
        win = hankel(str(1:W),str(W:end)); % all possible sliding windows

        myT = false(3,size(win,2));
        myT(1,:) = (win(cr,:) == double('i'))|(win(cr,:) == double('I'));
        myT(2,:) = (win(cr,:) == double('m'))|(win(cr,:) == double('M'));
        myT(3,:) = (win(cr,:) == double('o'))|(win(cr,:) == double('O'));
        allSeq(i).TNet = myT;



        myTSingle = zeros(1,size(win,2));
        myTSingle(win(cr,:) == double('i')) = 1;
        myTSingle(win(cr,:) == double('M')) = 2;
        myTSingle(win(cr,:) == double('o')) = 3;
        myTSingle(win(cr,:) == double('I')) = 1;
        myTSingle(win(cr,:) == double('m')) = 2;
        myTSingle(win(cr,:) == double('O')) = 3;
        allSeq(i).TSingle = myTSingle;
end
end
```

Figure 30 BinarizeTargets.m

The matrix |TNet| represents array targets for the neural network (1 -> [1 0 0], 2 -> [0, 1, 0], 3 -> [0 0 1]). The matrix |TSingle| contains just class (1,2,3).
(1 = 'i', 2 = 'M', 3 = 'o')

4.4.3.2 Input and target matrices construction

Once the input and target matrices are defined for each sequence, an input matrix |P|, a target matrix |TNet|, representing the encoding for all the sequences fed into the network, are created. The matrix |TSingle| is used as a target for all other types of classifiers.



```
% === construct input and target matrices
P = double([allSeq.P]); % input matrix
TNet = double([allSeq.TNet]); % target matrix
TSingle = [allSeq.TSingle]'; % target matrix
```

Figure 31 Construct input and target matrices

4.5 Summary

This chapter presented the list of datasets available and the data preparation techniques used for the implementations. Many transmembrane proteins databases serve as repositories for the sequences and structures of transmembrane proteins. Some databases such as OPM, PDB_TM contain orientations predictions of the protein relative to the membrane based on water-lipid transfer energy minimisation or hydrophobicity/structural feature analysis. OPM provides N-terminus localisation information. TOPDB was selected for the implementations. It includes TM proteins of unknown 3D structures whose topologies have been experimentally validated using low-resolution techniques such as gene fusion, antibody, and mutagenesis studies. BOCTOPUS 2 dataset was also selected, and it contains 42 proteins. This chapter also discussed the data preparation steps and techniques used for the implementations. Data preparation is the process of cleaning and transforming raw data before processing and analysis. This step was necessary for the implementations, and it involved reformatting data, making minor corrections to data and combing files. It was a lengthy process, but it was essential to put data in context and turn it into insights and eliminate bias resulting from poor data quality.

5. Implementation of ANN, KNN, and SVM

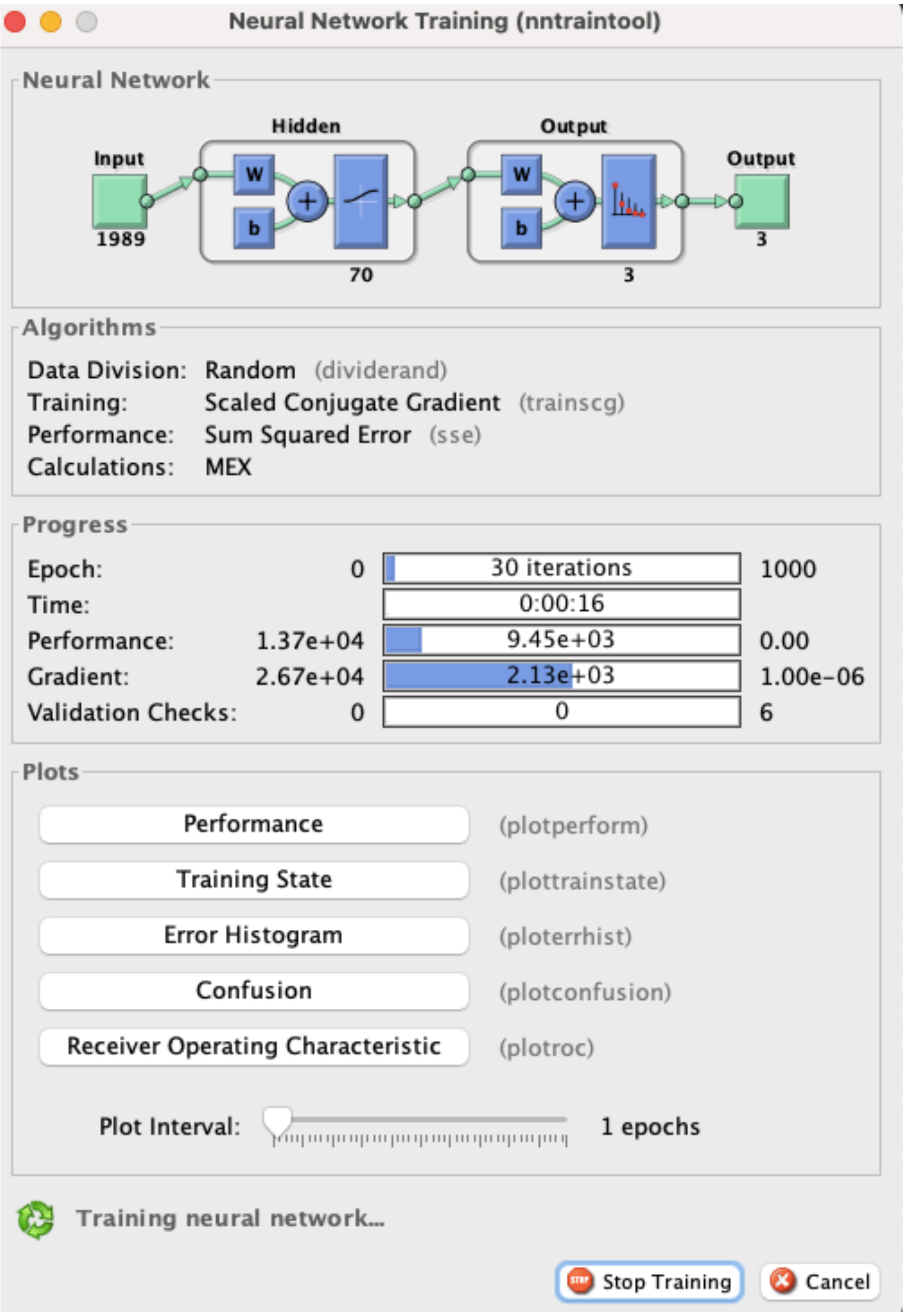
This chapter describes the implementations of ANN, KNN and SVM in MATLAB.

5.1 Implementation of ANN

5.1.1 Model building and training

A neural network with one input layer, one hidden layer, and one output layer is defined in the current architecture. The input layer encodes a sliding window in each input amino acid sequence, and a prediction is made on the structural state of the central residue in the window. A window of size 17 is chosen as a start. It is based on the statistical correlation between the secondary structure of a given residue position and the eight residues on either side of the prediction point. Each window position is encoded using a binary array of size 20, having one element for each amino acid type. In each group of 20 inputs, the element corresponding to the amino acid type in the given position is set to 1, while all other inputs are set to 0. Thus, the input layer consists of R = 17x20 input units, i.e., 17 groups of 20 inputs.

The output layer has three units. The individual unit corresponds to each topology. A binary scheme is used for encoding. The topologies localisations of all combinations in the sliding window are obtained to create the target matrix. The next step is to review the position in the centre of each window and the corresponding topology localisation using binary encoding as 1 0 0 (Outside: extracellular), 0 1 0 (Bacterial outer membrane) and 0 0 1(Inside: Periplasmic space). `nntraintool` opens the neural network training GUI. This function can be called to make the training GUI visible before training has occurred, after training if the window has been closed. It shows a representation of the layers of the network.



Algorithms

Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Sum Squared Error (sse)
Calculations: MEX

Progress

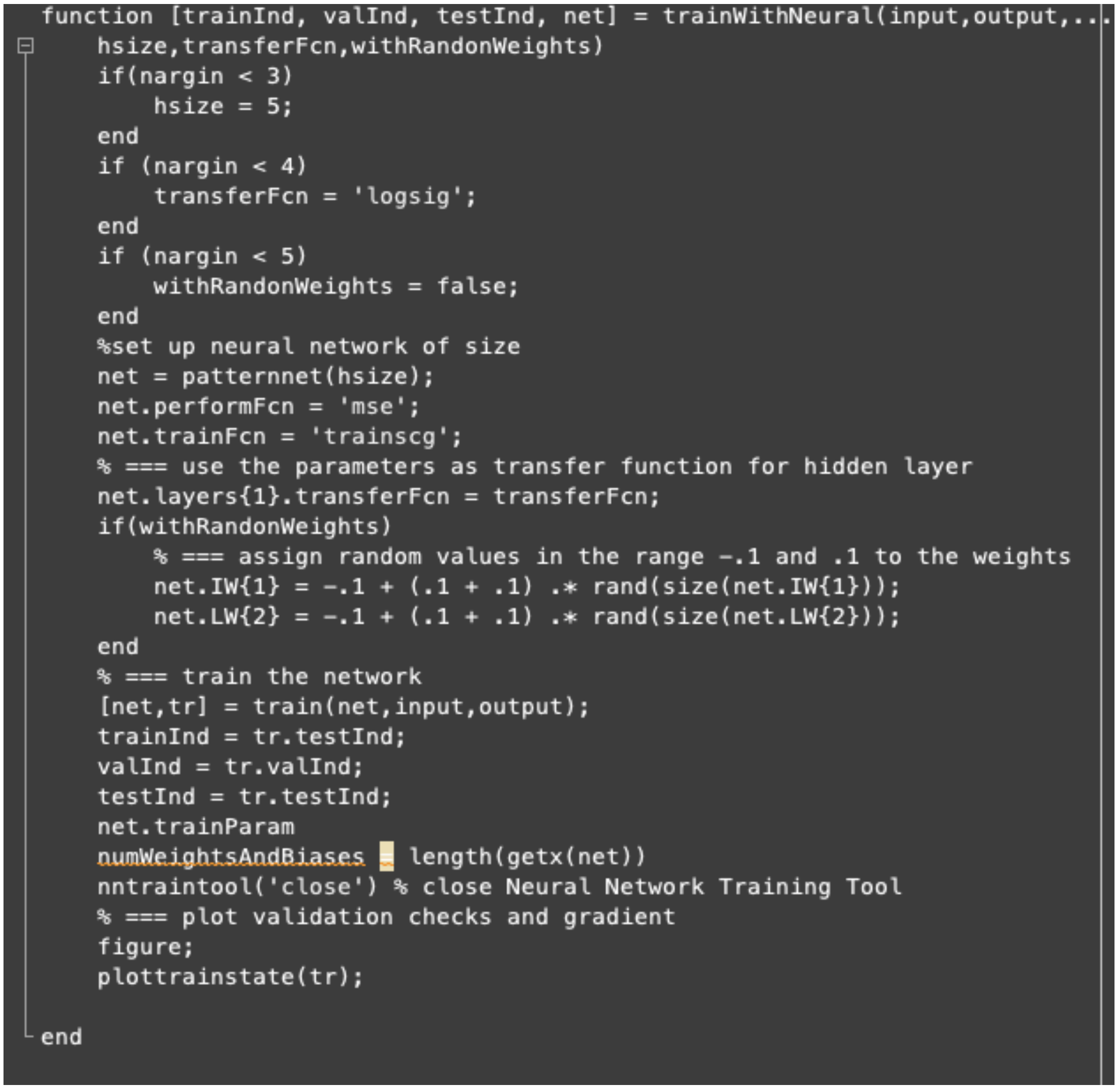
Epoch:	0	30 iterations	1000
Time:		0:00:16	
Performance:	1.37e+04	9.45e+03	0.00
Gradient:	2.67e+04	2.13e+03	1.00e-06
Validation Checks:	0	0	6

Plot Interval: 1 epochs

Training neural network...

Figure 32 Neural network training GUI

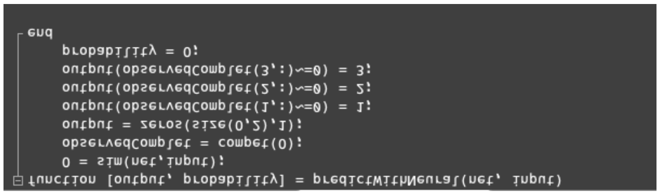
The function `trainWithNeural` is created and used to train the model.



```
function [trainInd, valInd, testInd, net] = trainWithNeural(input,output,...
    hsize,transferFcn,withRandonWeights)
    if(nargin < 3)
        hsize = 5;
    end
    if (nargin < 4)
        transferFcn = 'logsig';
    end
    if (nargin < 5)
        withRandonWeights = false;
    end
    %set up neural network of size
    net = patternnet(hsize);
    net.performFcn = 'mse';
    net.trainFcn = 'trainscg';
    % === use the parameters as transfer function for hidden layer
    net.layers{1}.transferFcn = transferFcn;
    if(withRandonWeights)
        % === assign random values in the range -.1 and .1 to the weights
        net.IW{1} = -.1 + (.1 + .1) .* rand(size(net.IW{1}));
        net.LW{2} = -.1 + (.1 + .1) .* rand(size(net.LW{2}));
    end
    % === train the network
    [net,tr] = train(net,input,output);
    trainInd = tr.testInd;
    valInd = tr.valInd;
    testInd = tr.testInd;
    net.trainParam
    numWeightsAndBiases = length(getx(net))
    nntraintool('close') % close Neural Network Training Tool
    % === plot validation checks and gradient
    figure;
    plottrainstate(tr);

end
```

Figure 33 Function `trainWithNeural`

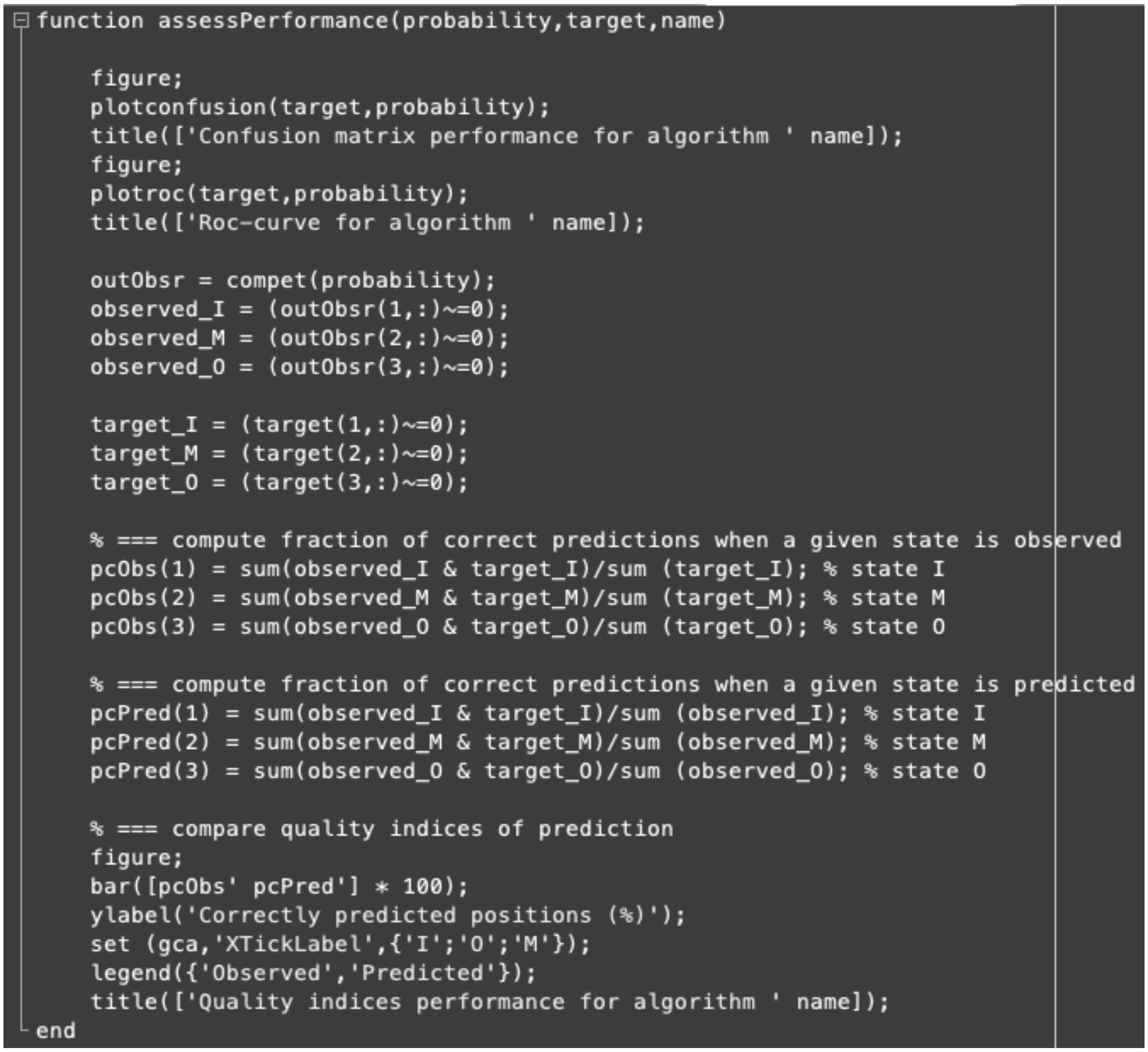
The function `predictWithNeural` returns probabilities of element being assigned to a particular class and the output is in the form of numbers 1,2,3 (1 = 'i', 2 = 'M', 3 = 'o').



```
function [output, probability] = predictWithNeural(net, input)
    o = sim(net,input);
    observedComplet = compet(o);
    output = zeros(size(o,2),1);
    output(observedComplet(1,:)~=0) = 1;
    output(observedComplet(2,:)~=0) = 2;
    output(observedComplet(3,:)~=0) = 3;
    probability = o;
end
```

Figure 34 Function `predictWithNeural`

The function `assessPerformance` is used to display performance in the form of ROC curves, confusion matrix and bar charts. `Plotconfusion (targets, outputs)` generates a confusion matrix for the target and output data. `Plotroc (targets, outputs)` plots the receiver operating characteristic for each output class. Bar (x, y) creates a bar graph. The main function used for the KNN implementation in MATLAB is named UseClassifiers.m.



```
function assessPerformance(probability,target,name)

    figure;
    plotconfusion(target,probability);
    title(['Confusion matrix performance for algorithm ' name]);
    figure;
    plotroc(target,probability);
    title(['Roc-curve for algorithm ' name]);

    outObsr = compet(probability);
    observed_I = (outObsr(1,:)~=0);
    observed_M = (outObsr(2,:)~=0);
    observed_O = (outObsr(3,:)~=0);

    target_I = (target(1,:)~=0);
    target_M = (target(2,:)~=0);
    target_O = (target(3,:)~=0);

    % === compute fraction of correct predictions when a given state is observed
    pcObs(1) = sum(observed_I & target_I)/sum (target_I); % state I
    pcObs(2) = sum(observed_M & target_M)/sum (target_M); % state M
    pcObs(3) = sum(observed_O & target_O)/sum (target_O); % state O

    % === compute fraction of correct predictions when a given state is predicted
    pcPred(1) = sum(observed_I & target_I)/sum (observed_I); % state I
    pcPred(2) = sum(observed_M & target_M)/sum (observed_M); % state M
    pcPred(3) = sum(observed_O & target_O)/sum (observed_O); % state O

    % === compare quality indices of prediction
    figure;
    bar([pcObs' pcPred'] * 100);
    ylabel('Correctly predicted positions (%)');
    set (gca,'XTickLabel',{'I';'O';'M'});
    legend({'Observed','Predicted'});
    title(['Quality indices performance for algorithm ' name]);
end
```

Figure 35 Function `assessPerformance`

5.1.2 Hyperparameters tuning

Various training algorithms have been used with the computation to evaluate prediction accuracy. Window size, bits encoding, transfer function, hidden layer size, training function, performance function and data division are all parameters that have been modified. All results can be found in Appendix A.

The neural network allows to set the following parameters:

- `hSize` – the size of the hidden layer
- `transferFcn` - transfer function for hidden layer
- `withRandonWeights` – whether there is a need to initialise network with random weights.

5.1.2.1 Hidden layer

TMB topology prediction can be classified as a pattern recognition problem. The network will be trained to recognise the topology of the residue in the centre of the sliding window based on other residues discovered in the sliding window. `patternnet` is used to create a pattern recognition neural network, and hsize corresponds to the size of the hidden layer. Various hidden layer sizes have been used during this computing from 2 to 1000. The accuracy of prediction is summarised in Appendix A.

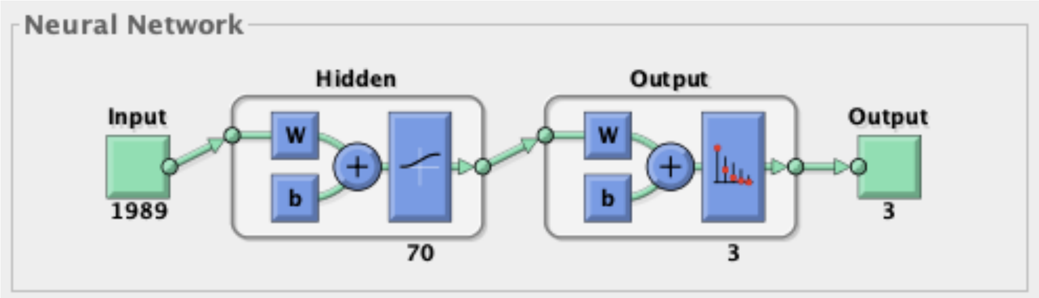


Figure 36 Representation of the neural network

5.1.2.2 Transfer function

During the implementation, various transfer functions have been used. Examples include Log-Sigmoid function and Tan-Sigmoid function. A transfer function, such as `logsig`, allows the signals received from the input layer to be transformed in each hidden. The output signal generated will be between 0 and 1. The neural can adjust the weights so that the error between the desired and observed is at the minimum. The log-sigmoidal function is represented in figure 37.

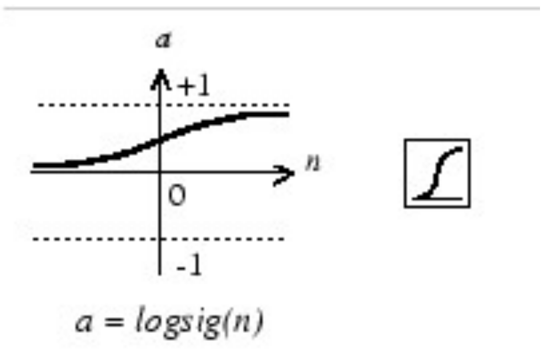


Figure 37 Log-sigmoid transfer function

Source: Srinivasu, G., Rao, R.N., Nandy, T.K. and Bhattacharjee, A. (2012)

A Log-Sigmoid function, which is also known as logistic function, is given by the following formula:

$$\sigma(t) = \frac{1}{1+e^{-\beta t}} \qquad (5.1.1)$$

β corresponds to the slope parameter. The sigmoid is like the step function; however, there is the addition of a region of uncertainty. Sigmoid functions are like the input-output relationships of biological neurons. Their derivatives are also easy to calculate, which helps calculate the weight updates for certain training algorithms. The derivative when β=1 is given by:

$$\frac{d\sigma(t)}{dt} = \sigma(t)[1 - \sigma(t)] \qquad (5.1.2)$$

When β ≠ 1, usi$\sigma(\beta, t) = \frac{1}{1+ e^{-\beta t}}$, the derivative is given by:

$$\frac{d\sigma(\beta,t)}{dt} = \beta[\sigma(\beta, t)[1 - \sigma(\beta, t)]] \qquad (5.1.3)$$

Another transfer function was used during the implementation of the ANN. It is named Hyperbolic tangent sigmoid transfer function. Tansig(N) takes one input and returns each element of N squashed between -1 and 1. Tansig is named after the hyperbolic tangent, which has the same shape. It is represented in figure 38.

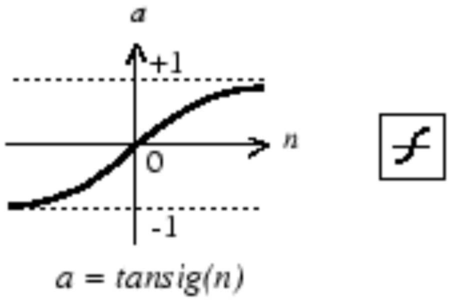


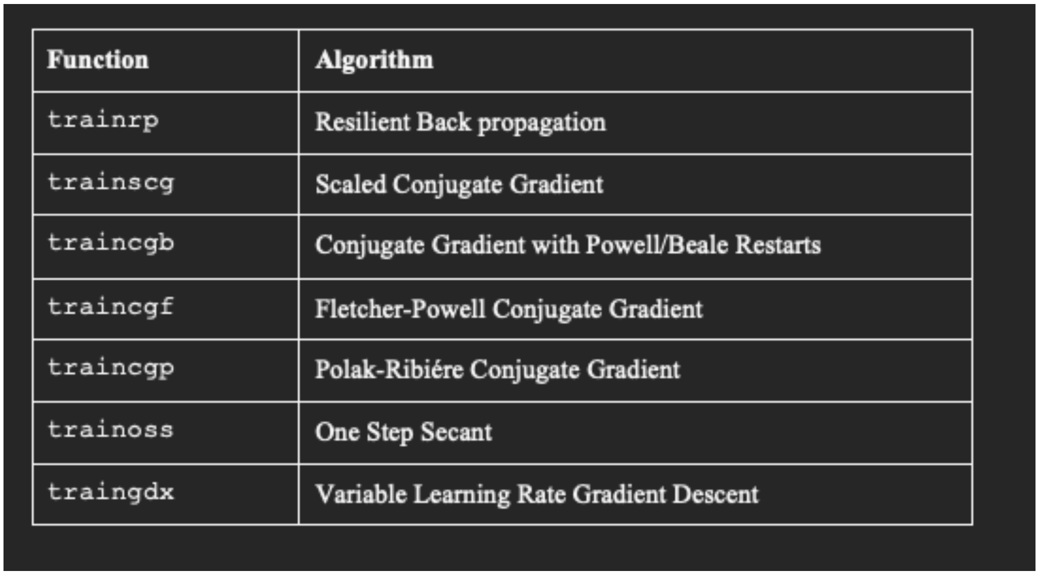
Figure 38 Tan-sigmoid transfer function

Source: Srinivasu, G., Rao, R.N., Nandy, T.K. and Bhattacharjee, A. (2012)

5.1.2.3 Training algorithms

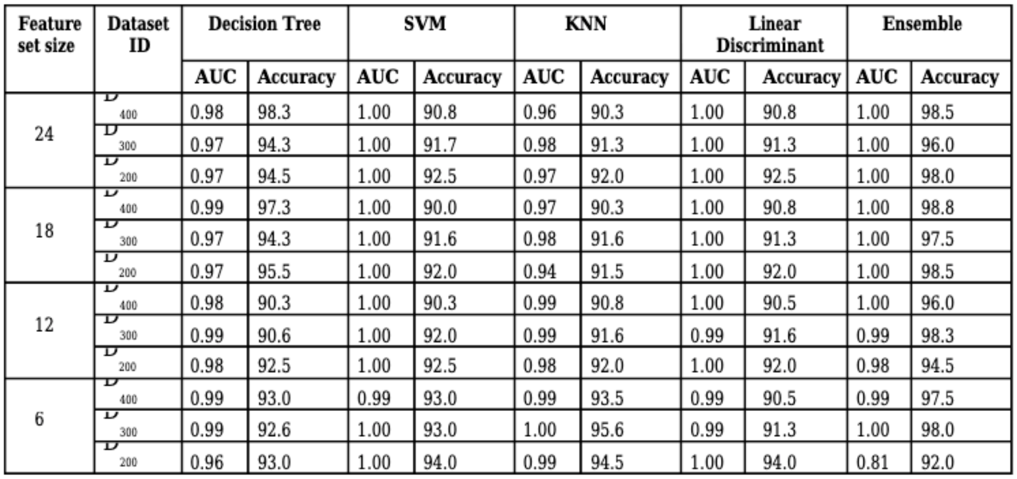
Various training algorithms have been used with the computation to evaluate prediction accuracy. A summary of the results with multiple training algorithms is found in Appendix A. A list of all training algorithms used is listed in table 4.

Table 4 Training algorithms used



Function	Algorithm
`trainrp`	Resilient Back propagation
`trainscg`	Scaled Conjugate Gradient
`traincgb`	Conjugate Gradient with Powell/Beale Restarts
`traincgf`	Fletcher-Powell Conjugate Gradient
`traincgp`	Polak-Ribiére Conjugate Gradient
`trainoss`	One Step Secant
`traingdx`	Variable Learning Rate Gradient Descent

`Trainscg` (scaled conjugate gradient) is the default training algorithm available in MATLAB. The training sequences are fed to the deep neural network, at each training cycle, through the sliding window described earlier, and this is done one residue at a time. SCG (Scaled Conjugate Gradient) (Møller, 1993) corresponds to a supervised learning algorithm for feed-forward neural networks. It is a member of the class of conjugate gradient methods (CGM). SCG is faster than standard backpropagation and other CGMs (Møller, 1993). The network training function `Trainrp` updates weight and bias values based on the resilient backpropagation algorithm. The network training function `Traincgb` updates weight and bias values based on the conjugate gradient backpropagation with Powell-Beale restarts. As part of all conjugate gradient algorithms, there is a periodic reset of the search direction to the negative of the gradient. The standard reset point corresponds when the number of iterations equals the number of network parameters (weights/biases). Other reset techniques can improve the efficiency of training. One method has been suggested by Powell (Powell, 1977) based on a previous technique described in Beale paper (Beale, 1972). The conjugate gradient backpropagation updates the weight and bias values with Fletcher-Reeves updates (Fletcher and Reeves, 1964) in the `traincgf` training function. Weight and bias values are updated according to the conjugate gradient backpropagation with Polak-Ribiére updates in `traincgp`. `Traincgp` has comparable performance to `traincgf`. In the `trainoss` training function, weight and bias values are updated based on the one-step secant method. The performance of various algorithms was evaluated in a recent article (Sharma et al., 2016) using small datasets with multiple dimensionalities. Their article indicated that KNN (k-nearest neighbors), SVM (support vector machines) and linear discriminant are the algorithms when using small datasets. A representation of the AUC and accuracies can be found in figure 39. Based on this study, it was decided to implement KNN and SVM classifiers for the topology prediction of TMB proteins.



Feature set size	Dataset ID	Decision Tree		SVM		KNN		Linear Discriminant		Ensemble	
		AUC	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC	Accuracy
24	D_{400}	0.98	98.3	1.00	90.8	0.96	90.3	1.00	90.8	1.00	98.5
	D_{300}	0.97	94.3	1.00	91.7	0.98	91.3	1.00	91.3	1.00	96.0
	D_{200}	0.97	94.5	1.00	92.5	0.97	92.0	1.00	92.5	1.00	98.0
18	D_{400}	0.99	97.3	1.00	90.0	0.97	90.3	1.00	90.8	1.00	98.8
	D_{300}	0.97	94.3	1.00	91.6	0.98	91.6	1.00	91.3	1.00	97.5
	D_{200}	0.97	95.5	1.00	92.0	0.94	91.5	1.00	92.0	1.00	98.5
12	D_{400}	0.98	90.3	1.00	90.3	0.99	90.8	1.00	90.5	1.00	96.0
	D_{300}	0.99	90.6	1.00	92.0	0.99	91.6	0.99	91.6	0.99	98.3
	D_{200}	0.98	92.5	1.00	92.5	0.98	92.0	1.00	92.0	0.98	94.5
6	D_{400}	0.99	93.0	0.99	93.0	0.99	93.5	0.99	90.5	0.99	97.5
	D_{300}	0.99	92.6	1.00	93.0	1.00	95.6	0.99	91.3	1.00	98.0
	D_{200}	0.96	93.0	1.00	94.0	0.99	94.5	1.00	94.0	0.81	92.0

Figure 39 AUC & Predictive Accuracy Value

Source: Sharma, S., Sharma, V. (2016)

5.1.2.4 Data division

Overfitting is a recurrent problem that occurs when the neural network is trained. This happens when the model is complex and the size of the training data is too small. There is memorisation of the network's training examples, but there is no generalisation. Early stopping is one of the methods to ensure generalisation. This default value in MATLAB for early stopping is a maximum of 6 iterations. In the early stopping method, data division creates three different sets. The first set is referred to as the training set. An ensemble of examples is used to learn, and the network weights are updated. The second set is the validation set, which is mostly used for tuning the model's parameters. During the initial training phase, the validation error habitually decreases (as well as the training set error). The error on the validation set typically starts to rise when the network begins overfitting the data. The network weights and biases are saved when the validation set error reaches its minimum. The third set is referred to as the test set. It is used to confirm the predictive power of the network. Four functions are available in MATLAB for data division. The first one is `dividerand`. It will divide the data randomly. The second one is `divideblock`. It will divide the data into contiguous blocks. The third one is `divideint`. The data will be divided using an interleaved selection. The last one is `divideind`. Data division happens automatically during the training of the network. Data is divided into three sets: the training set, validation set, and testing set. Various division functions were used, and the results were evaluated. When `dividerand` is used, the data is divided into three sets randomly. The division parameter `net.divideParam.trainRatio`, `net.divideParam.valRatio`, and `net.divideParam.testRatio` are used. This is the default setting. The ratio that is used by default is 0.7/0.15/0.15. It corresponds to the ratio for training, testing and validation. Other types of partitioning can be used when modifying `net.divideFcn`. Parameters of the division function are stored and can be modified with the property: `net.divideParam`. When using the function train, by default, the data is randomly divided so that 70% of the samples are assigned to the training set, 15% to the validation set, and 15% to the test set. Figures 40,41, and 42 show the structural assignments in the training, validation and test datasets used for this network where 'I' corresponds to inner-loop, 'O' corresponds to outer-loop, and 'M' corresponds to membrane lipid-facing and membrane pore-facing.

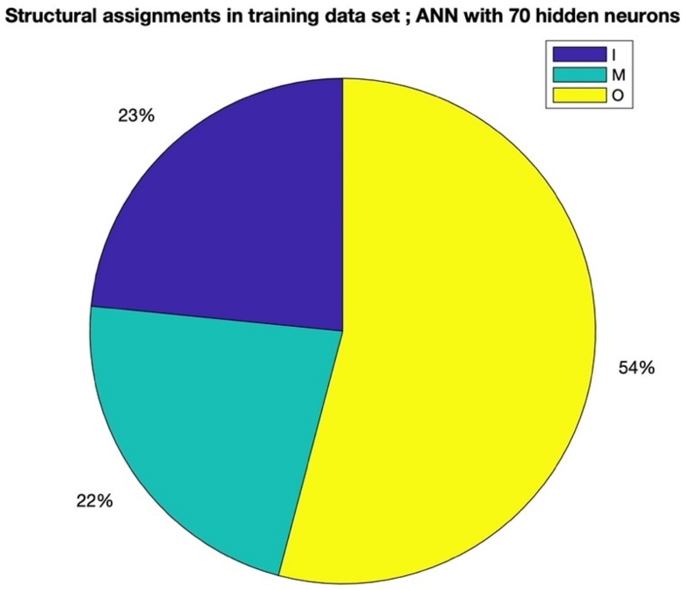


Figure 40 Structural assignment in the training data set

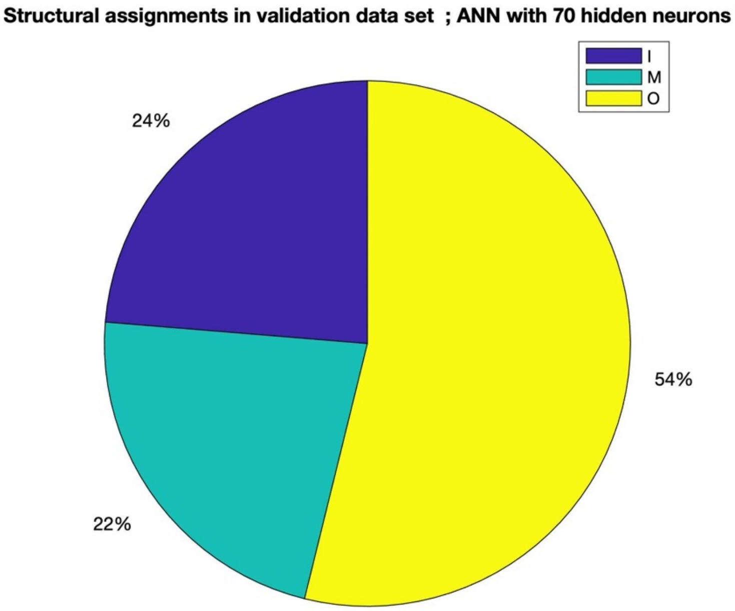


Figure 41 Structural assignment in the validation data set

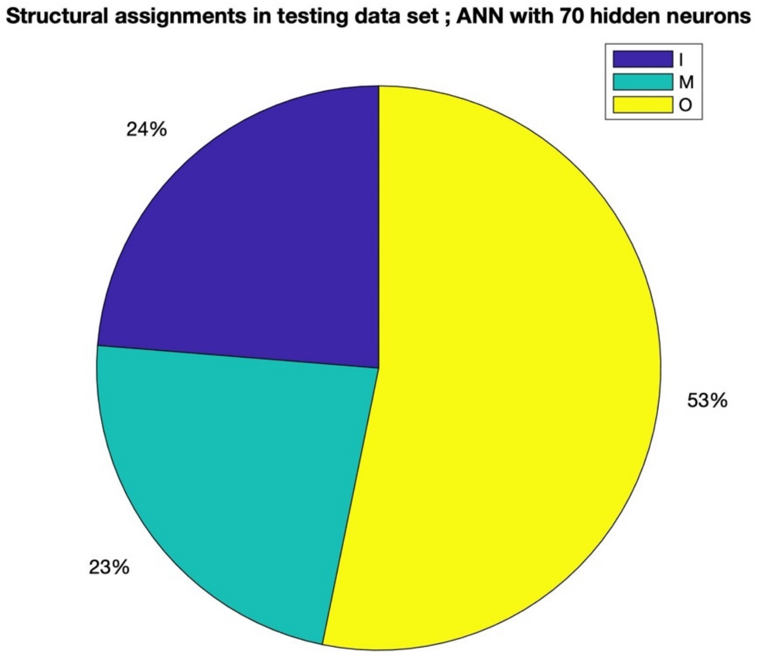
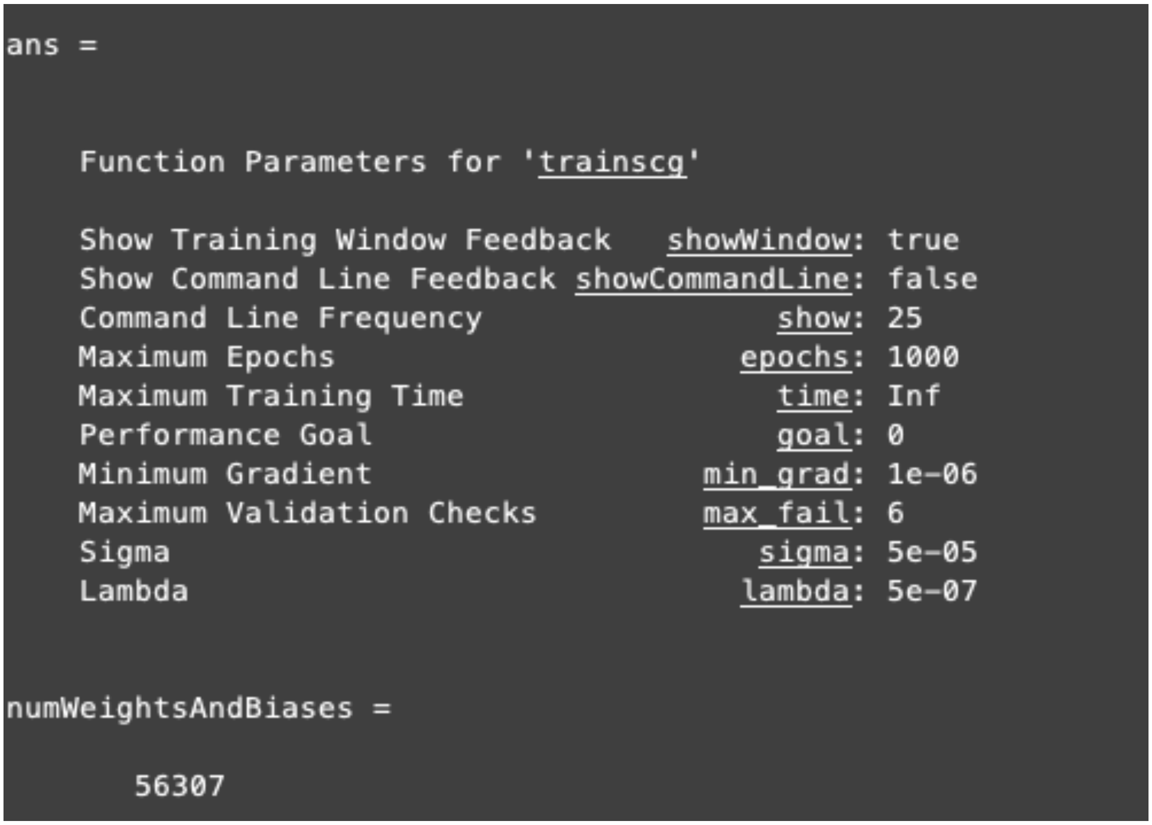


Figure 42 Structural assignment in the testing data set

The topology of the residues in the three subsets is similar when comparing pie charts in figures 40, 41 and 42. There is a measure of error for neural network training between computed outputs and desired target outputs of the training data. Mean Squared Error is a common measure. The mean squared error (MSE) of an estimator calculates the average of the squares of the “errors” in statistics. Error is the difference between what is estimated and the estimator. MSE corresponds to a risk function based on the expected value of the squared error loss. However, some research results (Golik, Doetsch & Ney, 2013) suggest using a different measure, called cross-entropy error, as it is sometimes preferable to using mean squared error. As discussed earlier, there is monitoring the error on the validation during the training phase. Training, validation, and test errors are displayed with the function `plotperform`.

Conditions are defined in the `net.trainParam`, and when they are met, the training stops. For example, the number of epochs greater than 200 can be a condition. Figure 43 represents the various functions parameters of trainscg, which was used for this project. Other training functions have been used.



```
ans =


    Function Parameters for 'trainscg'

    Show Training Window Feedback   showWindow: true
    Show Command Line Feedback showCommandLine: false
    Command Line Frequency                show: 25
    Maximum Epochs                      epochs: 1000
    Maximum Training Time                 time: Inf
    Performance Goal                      goal: 0
    Minimum Gradient                  min_grad: 1e-06
    Maximum Validation Checks         max_fail: 6
    Sigma                                sigma: 5e-05
    Lambda                              lambda: 5e-07


numWeightsAndBiases =

       56307
```

Figure 43 net.trainParam function results

For example, in the training considered, the training process stops when the validation error increases for a specified number of iterations (6) or the maximum number of allowed iterations is reached (1000). Validation checks represent the number of successive iterations that the validation performance fails to decrease. Figure 44 illustrates the magnitude of the gradient. Figure 45 represents the number of iterations. It can be seen clearly that validation checks=6 at epoch 158, and the training stops automatically. When the training has obtained a minimum performance, the gradient will be small when the gradient is less than 1e-6. For this project, the validation checks (>6) triggered the training to stop.

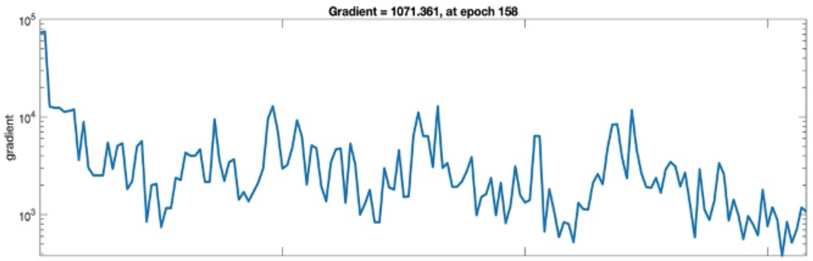


Figure 44 Gradient

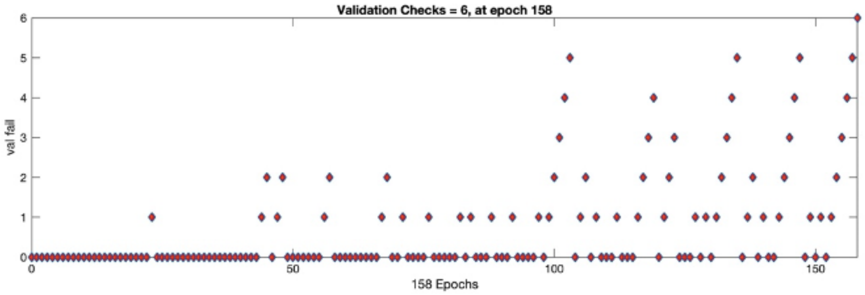


Figure 45 validation checks

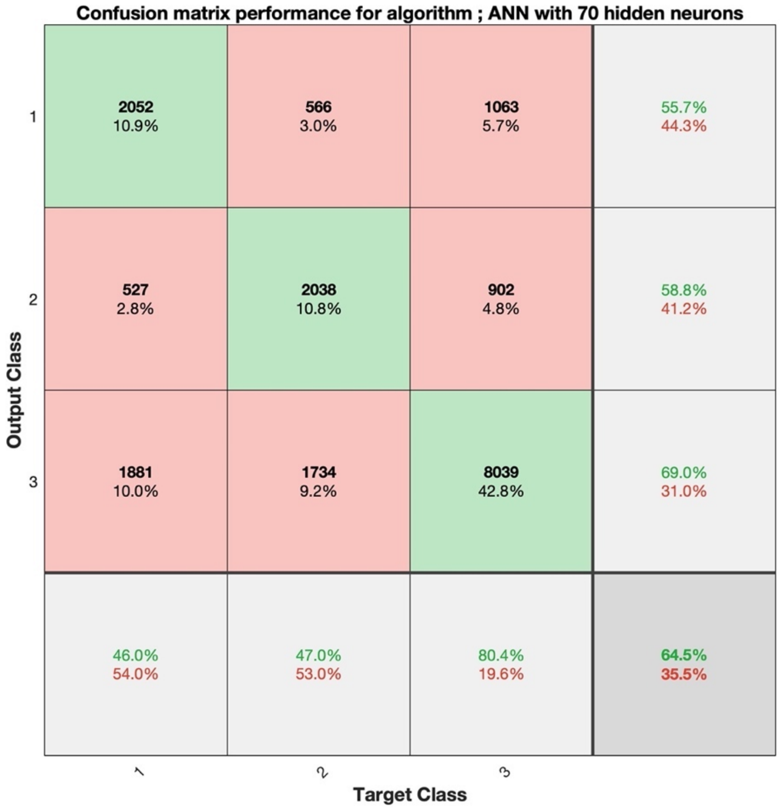
The magnitude of the gradient can be adjusted by changing the parameter `net.trainParam.min_grad`. The criterion of validation checks can be modified by changing the parameter `net.trainParam.max_fail`. Finally, stopping the training manually by clicking ‘stop training’ in the training window is also possible.

5.1.3 Performance evaluation

5.1.3.1 Confusion matrix

In machine learning, a confusion matrix, also known as a contingency table or an error matrix, is a specific table layout that allows visualisation of the performance of an algorithm, typically a supervised learning one (in unsupervised learning, it is usually called a matching matrix). Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class.

A review of the confusion matrix is needed to analyse the network response. It is essential to consider the outputs of the trained network and compare them with the expected results, also referred to as targets. In the training tool window, there is a button that permits displaying the confusion matrix. The confusion matrix is represented in figure 46. The diagonal cells show the number of residue positions correctly classified for each topology class. The off-diagonal cells show the number of residue positions that were misclassified. The diagonal cells correspond to observations that are correctly classified. The number of words and the percentage of the total number of observations is shown in each cell. The column on the far right of the plot shows the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. These metrics are often called the precision (or positive predictive value) and false discovery rate. The row at the bottom of the plot shows the percentages of all the examples belonging to each class that are correctly and incorrectly classified. These metrics are called the recall (or true positive rate) and false-negative rate. The cell in the bottom right of the plot shows the overall accuracy.



Confusion matrix performance for algorithm ; ANN with 70 hidden neurons

Output Class \ Target Class	1	2	3	
1	**2052** 10.9%	**566** 3.0%	**1063** 5.7%	55.7% 44.3%
2	**527** 2.8%	**2038** 10.8%	**902** 4.8%	58.8% 41.2%
3	**1881** 10.0%	**1734** 9.2%	**8039** 42.8%	69.0% 31.0%
	46.0% 54.0%	47.0% 53.0%	80.4% 19.6%	**64.5%** **35.5%**

Figure 46 Confusion matrix

5.1.3.2 Receiver operating characteristic curve

The Receiver Operating Characteristic (ROC) curve can also be looked at. It is a plot of the true positive rate (sensitivity) versus the false-positive rate (1 – specificity). True positive rate is also known as sensitivity in biomedical engineering. Sensitivity corresponds to the proportion of actual positives that are correctly identified. False-positive corresponds to fall-out. Fall-out is related to the specificity and equals to 1 - specificity. The ROC curve represents the sensitivity as a function of the fall-out. A perfect predictor would be described at 100% sensitivity. The overall accuracy is better when the ROC curve is close to the upper left corner (100% sensitivity, 100% specificity).

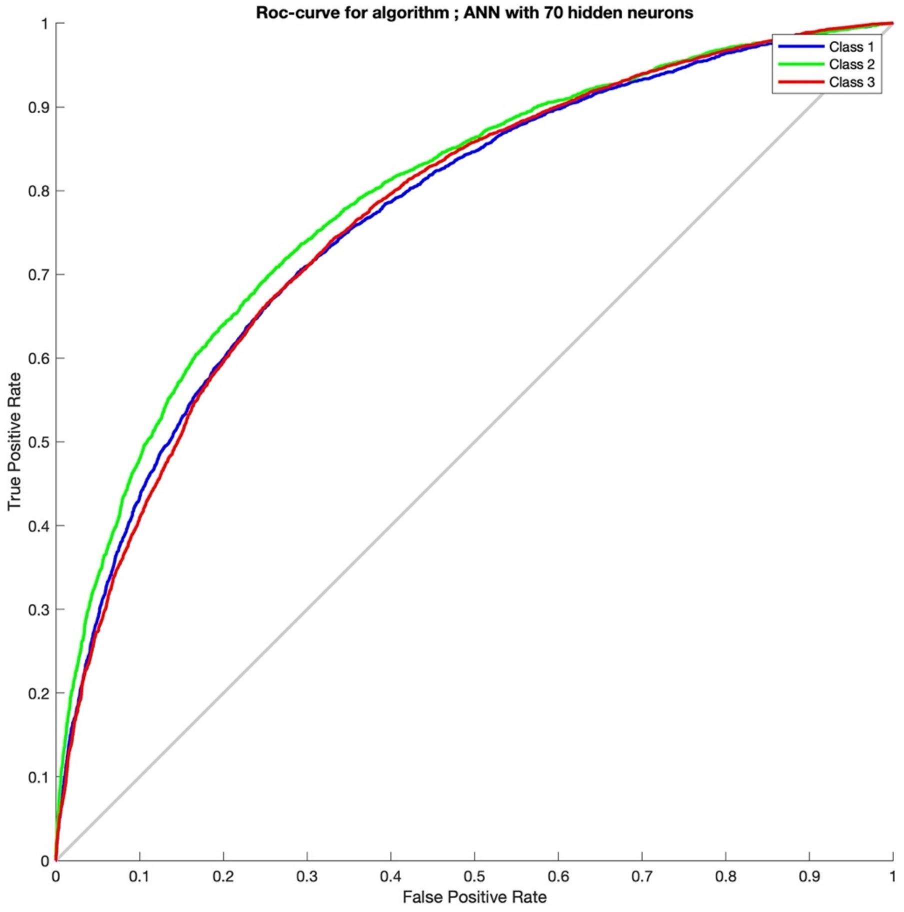


Figure 47 Receiver Operating Characteristics curve

5.1.3.3 Prediction quality indices

The topology prediction is evaluated in detail by calculating prediction quality indices represented in figure 48. They indicate how well a particular state is predicted and whether overprediction or underprediction has occurred. The index `pcObs(S)` was defined for state S (S = {I, O, M}) as the number of residues correctly predicted in state S, divided by the number of residues observed in state S. Similarly, the index `pcPred(S)` for state S was defined as the number of residues correctly predicted in state S, divided by the number of residues predicted in state S. These quality indices are useful for the interpretation of the prediction accuracy.

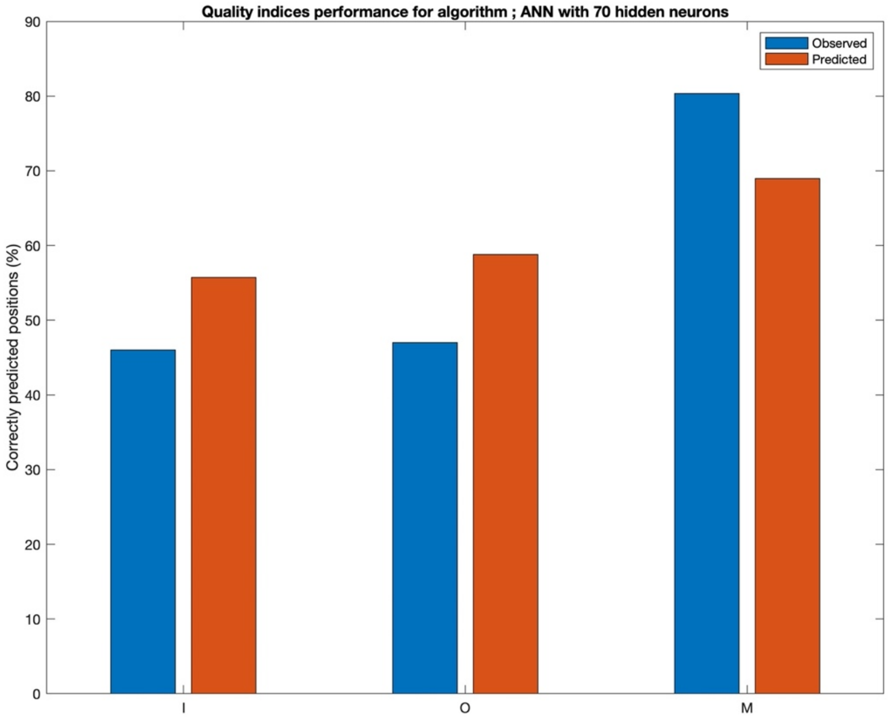


Figure 48 Quality indices of prediction

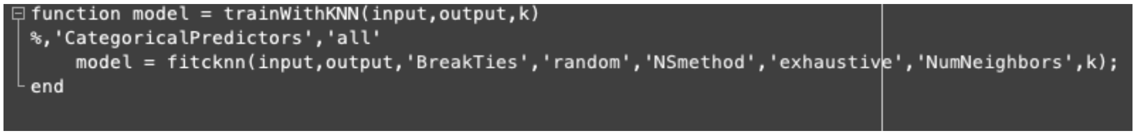
5.2 Implementation of KNN

Some elements used for the implementation of the ANN can be reused to implement a KNN (k-nearest neighbors) classifier. The dataset(s) and binarisation of the inputs used will be the same. The datasets used will be the same datasets used with the ANN implementation. The BOCTOPUS2 and TOPDB datasets will be curated and used for implementation.

5.2.1 Model building and training

The classifier uses the fitcknn function. fitcknn is part of the statistics toolbox. `Fitcknn (_, Name, Value)` creates a model with other options specified with one or more name-value pair arguments. The tie-breaking algorithm, distance metric of observation weights can be identified. Name-value pair arguments are represented as Name, Value arguments separated by a comma. Name is the argument name. Value is the corresponding value argument. The name must be within single quotes. An example is: `‘NumNeighbors',15,'NSMethod','exhaustive'`,'`Distance`','`euclidean`’ specifies a classifier for fifteen-nearest neighbors using the nearest neighbor search method and the Euclidean metric. Several model parameters can be modified, and there will be a first review on ‘`BreakTies`', '`NSMethod`' and '`NumNeighbors`' that have been used as a basis for creating the k-nearest neighbors in MATLAB. '`BreakTies`' corresponds to the tie-breaking algorithm. The default value is 'smallest’, but values can be 'nearest' or 'random'. predict method uses the tie-breaking algorithm. There are three ways to break a tie for a k-nearest neighbors classifier. The smallest index among tied groups is used with 'smallest'. The nearest neighbor among tied groups is used with 'nearest'. A random tiebreaker among tied groups is used with 'random'. 'random' is like flipping a coin and deciding positive/negative for example. There is a tie when various classes have the same number of nearest points among the k nearest neighbors. '`BreakTies`', 'nearest' is an example of the syntax used in the code. Ties happen when there is the same number of votes for two different classes in the set of nearest neighbors. In binary classification, it can occur if k is even. Changing to an odd value (such as 3 or 5) in binary cases is possible, but it does not help for multi-class. Missing values need to be dealt with when using nearest neighbors. An attribute has usually values in the data. Missing values happen when a testing instance does not have an attribute. Missing values are problematic for nearest neighbors. There is no trick, like for naive Bayes nearest neighbors classifier or for decision tree that helps to deal with missing values. Here, it is necessary to do something. It is required to fill in the missing values; otherwise the model can’t compute distance. A reasonable choice can be to choose the mean (average) value of the attribute across the dataset.

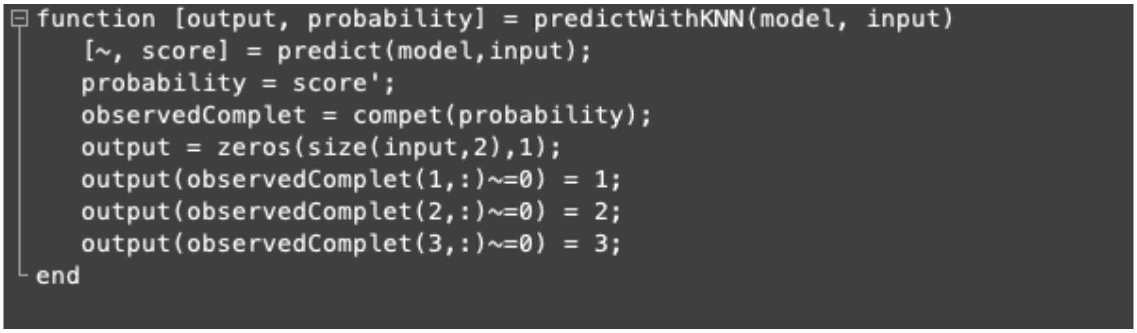
The function `trainWithKNN` is created and used to train the model.



```
function model = trainWithKNN(input,output,k)
%,'CategoricalPredictors','all'
    model = fitcknn(input,output,'BreakTies','random','NSmethod','exhaustive','NumNeighbors',k);
end
```

Figure 49 Function `trainWithKNN`

The function `predictWithKNN` returns probabilities of element being assigned to a particular class and the output is in the form of numbers 1,2,3 (1 = 'i', 2 = 'M', 3 = 'o').



```
function [output, probability] = predictWithKNN(model, input)
    [~, score] = predict(model,input);
    probability = score';
    observedComplet = compet(probability);
    output = zeros(size(input,2),1);
    output(observedComplet(1,:)~=0) = 1;
    output(observedComplet(2,:)~=0) = 2;
    output(observedComplet(3,:)~=0) = 3;
end
```

Figure 50 Function `predictWithKNN`

A model parameter that was discussed earlier is the tie-breaking algorithm. Another model parameter that needs to be considered is '`NSMethod`' corresponds to the Nearest neighbor search method. The nearest neighbor search method is coded as pair consisting of '`NSMethod`' and '`kdtree`' or 'exhaustive'. The default value is '`kdtree`'. A kd-tree is created, and find the nearest neighbors when the value 'kdtree' is used. The distance metric must be '`euclidean`', '`cityblock`', '`chebychev`' or '`minkowski`'in order for '`kdtree`' to be valid. The exhaustive search algorithm is used with '`exhaustive`'. There is a computation of the distance values from all points in X to each point in Y to find the nearest neighbors. An example of the syntax for this parameter is '`NSMethod`', 'exhaustive'. (Mathworks, 2016). A k-d tree is built from training data. An example of a kd-tree is represented in figure 51.

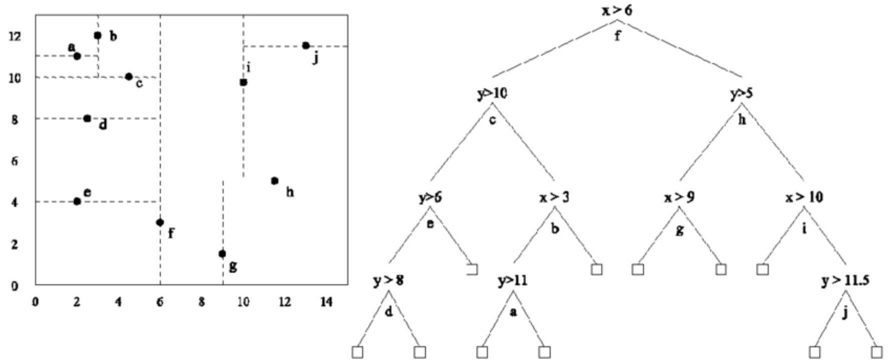


Figure 51 Scheme of a KD-Tree Search algorithm

Source: Kraus, P. and Dzwinel, W. (2012)

In a k-d tree, a data structure is built and organises the dataset as a tree. To find the nearest neighbors to the query point, it is necessary to navigate down the tree to the region that will hopefully contain most of the nearest neighbors for the testing point. The algorithm works by picking a random attribute, and for that attribute, it finds the median. It uses that median to split the dataset. It will divide it evenly with half the data points on one side and half of the data points on the other side. The procedure is repeated at other iterations using a different random attribute. The procedure is repeated until it ends up with a predetermined number of points left in each branch of the tree. Once you know the number of points that will be, you know how deep the tree is. At each level, the dataset is split in half, and because the median is selected, it is split exactly in half. The dataset gets twice a small with every step. The previous figure states that the classifier stops when there are 2 or 3 nodes in each leaf, and figure 51 represents the final cutting of the space into regions. The data structure is then used for finding nearest neighbors to testing queries. In this example, for finding the nearest neighbors for a new point (7,4), the region containing (7,4) is selected, and a comparison to all points in the region is done. This technique can easily miss real nearest neighbors. '`NumNeighbors`' is another model parameter used to create the k-nearest neighbors classifier in MATLAB in this paper. When the number of neighbors is changed to 3 or 4, the model will classify using the third or fourth nearest neighbors. K=2 has been selected to start the process. Values will be modified later to obtain better results and optimise the classifier. k will affect the algorithm's performance a lot, and it is discussed using an example in another section of this research.

So far, the `trainwithKNN` function has been discussed. `PredictwithKNN` function will return probabilities of the element being assigned to the class and the output in the form of numbers 1,2,3 (1= 'i', 2='M', 3='o') based on the trained-nearest neighbor classification model. The classifier uses the predict function available in MATLAB. The syntax is as follow: `[output, probability] = predictwithKNN (model, input)`. As discussed earlier, a k-nearest neighbors classification model is defined as a Model and specified as a ClassificationKNN model object returned by fitcKNN. Input corresponds to the predictor data to be classified.

The function `assessPerformance` is used to display performance in the form of ROC curves, confusion matrix and bar charts.

5.2.2 Hyperparameters tuning

The k-NN model uses a k value=2 as a first implementation. Variations in the k values will be used to evaluate the performance of k-NN, and all results are summarised in Appendix B. Multiple parameters of a KNN are modifiable. `NumNeighbors` is one of the properties of a KNN that has been modified. This positive number corresponds to the number of nearest neighbors in X, which is a numeric matrix of unstandardised predictor value. One predictor (variable) is represented by each column of X. One observation is represented by each row. For example, changing the neighborhood size to 6 means that the model classifies using the six nearest neighbors. A single nearest neighbor is the default k-nearest neighbors classifier. Using more than one neighbor is better `NSMethod` and `BreakTies` are other properties of a KNN that have been modified.

The implementation used in MATLAB for this project runs 1-18 using the 'exhaustive' nearest neighbors search method, `BreakTies`, '`random`' and varied data division for training. The syntax is as follow in the `trainWithKNN.m` file. Runs 19-33 use a k-d tree search method with various distance metrics. The Tie-breaking algorithm remains the same (random) as the one used in runs 1-18. '`euclidean`' corresponds to the Euclidean distance. '`cityblock`' corresponds to the City block distance, '`minkowski`' is the `Minkoswki` distance. 2 is the default exponent. '`chebyshev'` corresponds to the Chevychev distance (maximum coordinate difference).

For runs 34-52, the tie-breaking algorithm has been modified. All runs used for far were based on a random tie-breaking algorithm. The tie-breaking algorithm has been changed to `'smallest'` or `'nearest'`. '`BreakTies`' applies when '`includeTies`' is false. '`includeTies`' is a flag that can be true or false. It is false by default. When it is set to 'true', the model considers all nearest neighbors with a distance equal to the kth smallest distance in the output arguments. If it is false, then the model chooses the observation with the smallest index among the observations that have the same distance from a query point. '`Bucketsize`' can also be modified. It corresponds to the maximum data points in node.50 is the default. It is used only when '`NSMethod`' is '`kdtree`'. The code corresponds to s a pair of '`Bucketsize`' and a positive integer value separated by a comma. An example is `'Bucketsize',40`. `'exponent'` is the Minkowski distance exponent, and it can be added to one of the sub-parameters of a KNN. It is only applicable when `'Distance'` is specified as '`minkoswki`'.

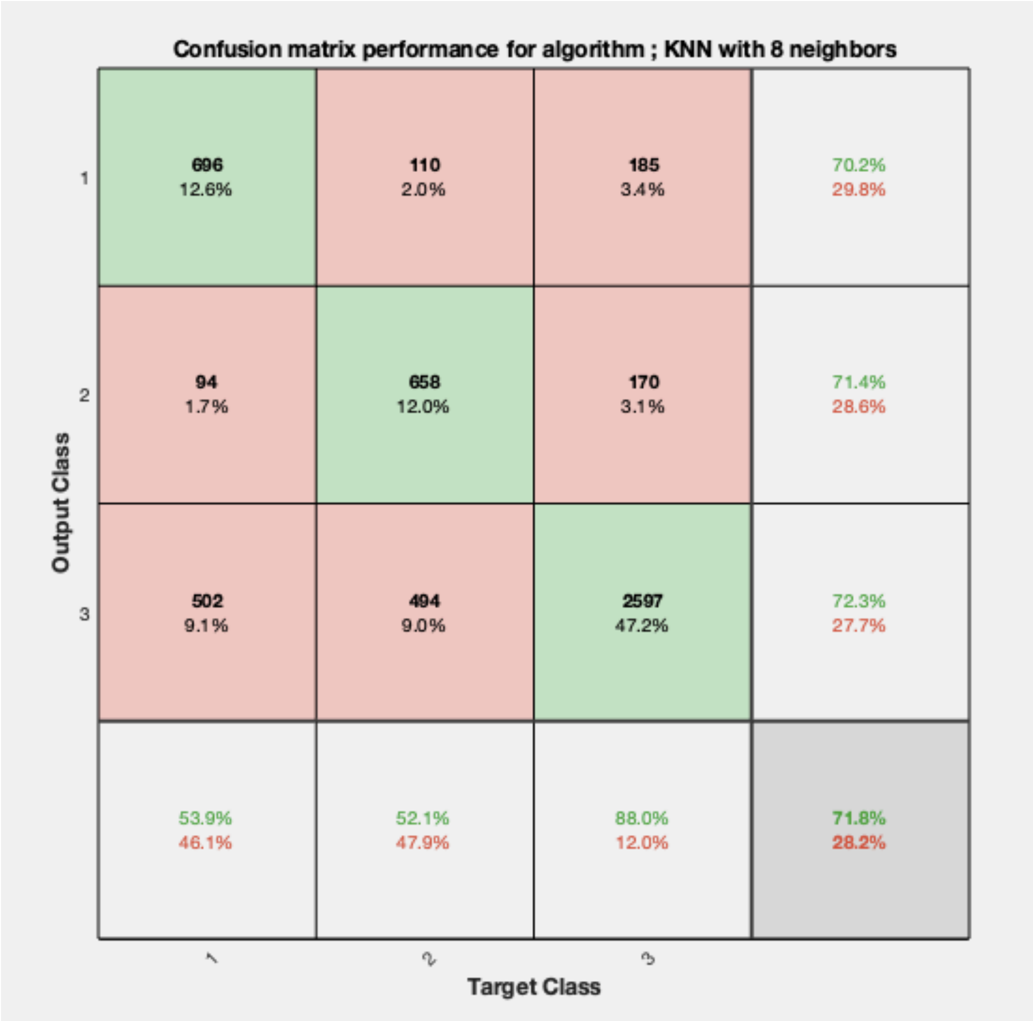
Starting from run 53, all the runs use the dataset `TOPBPdataset.mat`. Parameters are adjusted similarly to the runs 1-52, and results can be found in Appendix B.

5.2.3 Performance evaluation

With the implementation of KNN, plots are also provided to evaluate the performance of the KNN classifier. Plots include a confusion matrix, Receiver Operating Characteristic (ROC) curve and chart with predicted and observed positions. The description of the plots and interpretation of results have been already defined in the previous chapter.

5.2.3.1 Confusion matrix

The confusion matrix is represented in figure 52. The best overall accuracy is 71.8%. The diagonal cells show the number of residue positions correctly classified for each topology class. The off-diagonal cells show the number of residue positions that were misclassified. The diagonal cells correspond to observations that are correctly classified. The number of observations and the percentage of the total number of observations is shown in each cell. The column on the far right of the plot shows the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. These metrics are often called the precision (or positive predictive value) and false discovery rate. The row at the bottom of the plot shows the percentages of all the examples belonging to each class that are correctly and incorrectly classified. These metrics are called the recall (or true positive rate) and false-negative rate. The cell in the bottom right of the plot shows the overall accuracy.



Confusion matrix performance for algorithm ; KNN with 8 neighbors

Output Class \ Target Class	1	2	3	
1	696 12.6%	110 2.0%	185 3.4%	70.2% 29.8%
2	94 1.7%	658 12.0%	170 3.1%	71.4% 28.6%
3	502 9.1%	494 9.0%	2597 47.2%	72.3% 27.7%
	53.9% 46.1%	52.1% 47.9%	88.0% 12.0%	71.8% 28.2%

Figure 52 Confusion matrix

5.2.3.2 Receiver operating characteristic curve

The Receiver Operating Characteristic (ROC) curve can also be looked at. It is a plot of the true positive rate (sensitivity) versus the false positive rate (1 – specificity). True positive rate is also known as sensitivity in biomedical engineering. Sensitivity corresponds to the proportion of actual positives that are correctly identified. False-positive corresponds to fall-out. Fall-out is related to the specificity and equals to 1 - specificity. The ROC curve represents the sensitivity as a function of the fall-out. A perfect predictor would be described at 100% sensitivity. When the ROC curve is close to the upper left corner (100% sensitivity, 100% specificity), the overall accuracy is better.

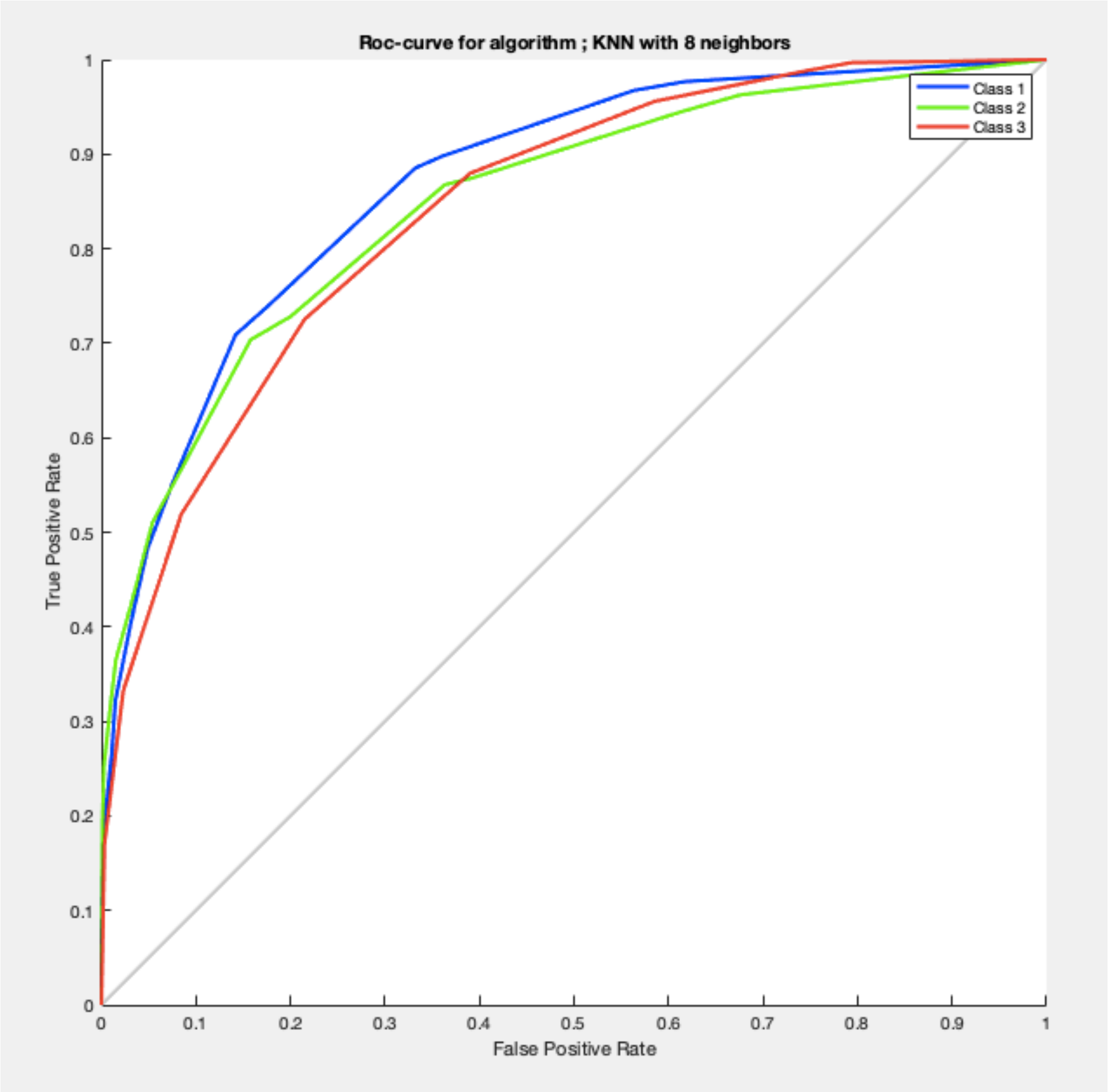


Figure 53 ROC curve

5.2.3.3 Prediction quality indices

The topology prediction is evaluated in detail by calculating prediction quality indices represented in figure 54. They indicate how well a particular state is predicted and whether overprediction or underprediction has occurred. The index `pcObs(S)` was defined for state S (S = {I, O, M}) as the number of residues correctly predicted in state S, divided by the number of residues observed in state S. Similarly, the index `pcPred(S)` for state S was defined as the number of residues correctly predicted in state S, divided by the number of residues predicted in state S. These quality indices are useful for the interpretation of the prediction accuracy.

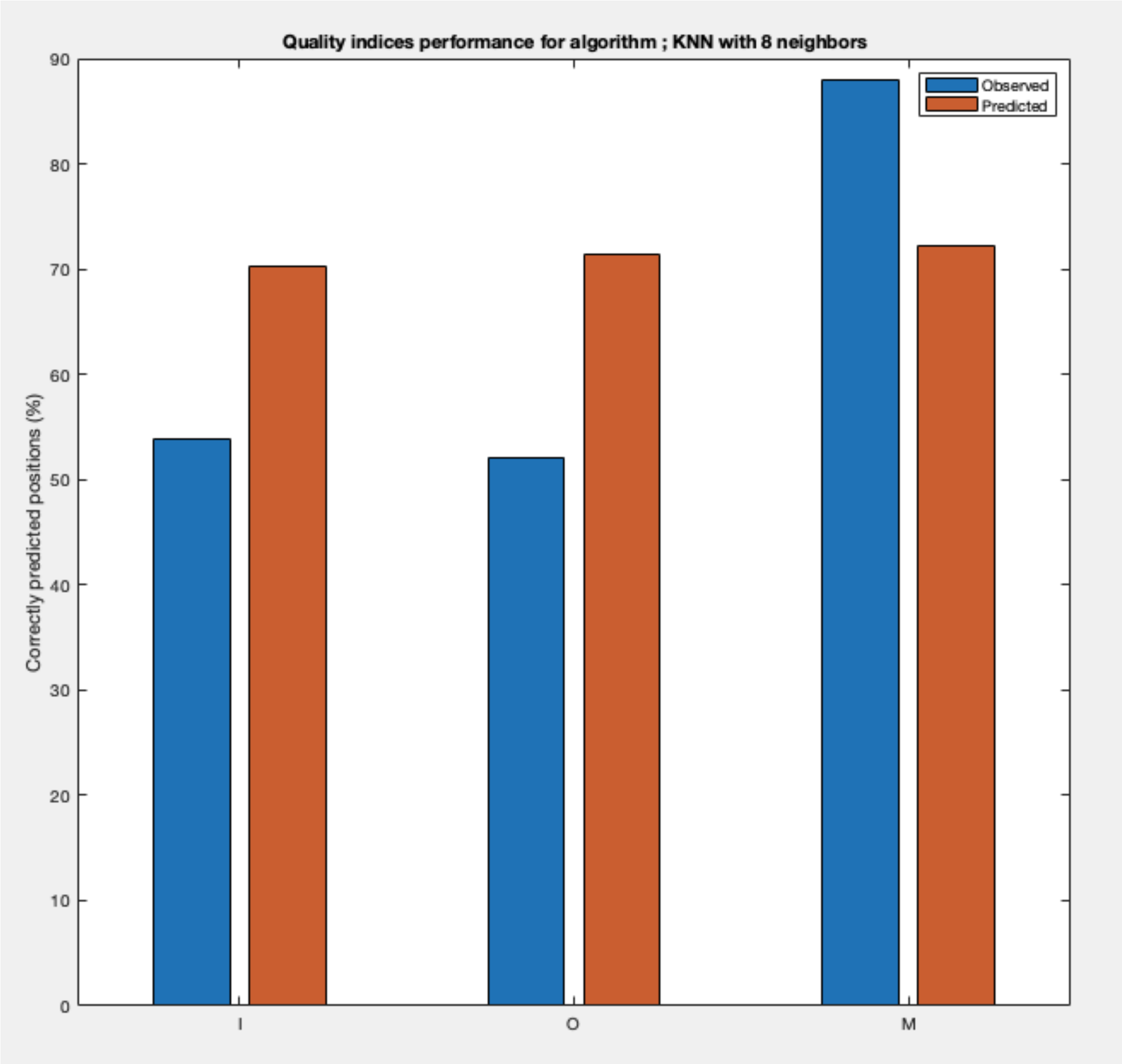


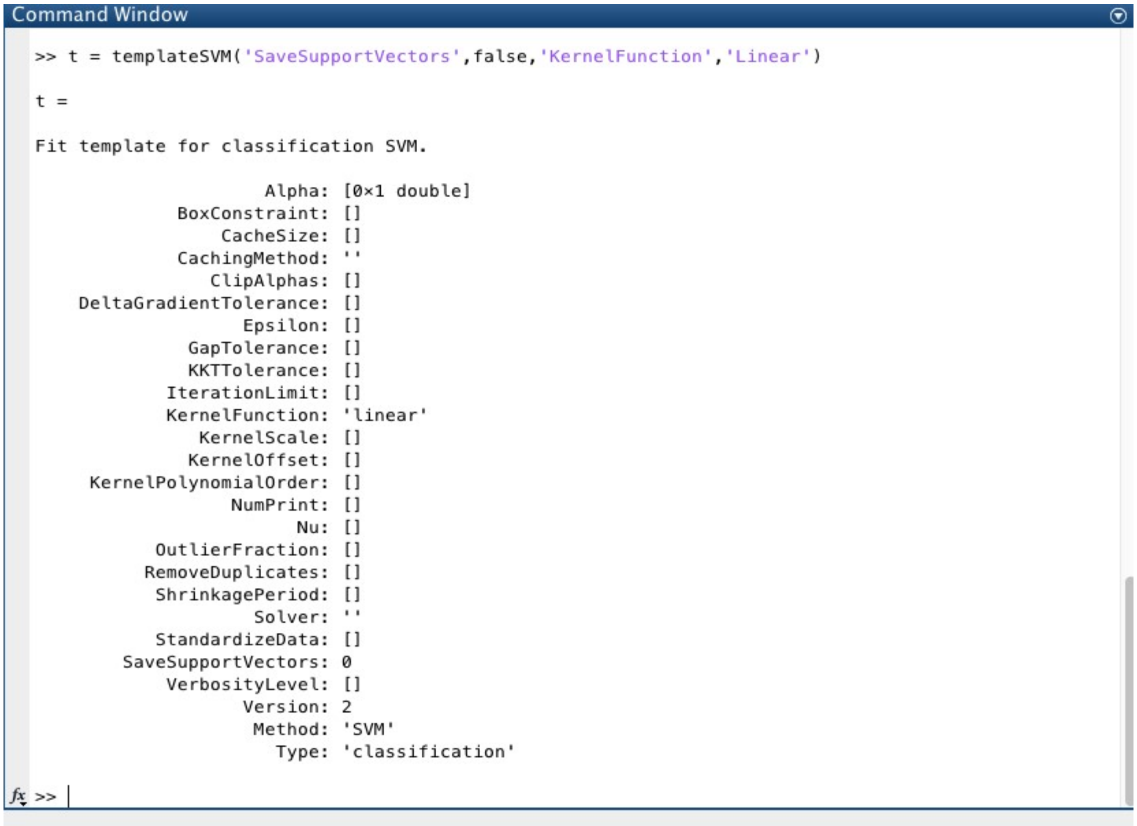
Figure 54 Quality indices

5.3 Implementation of SVM

This section describes the implementation of SVM in MATLAB.

5.3.1 Model building and training

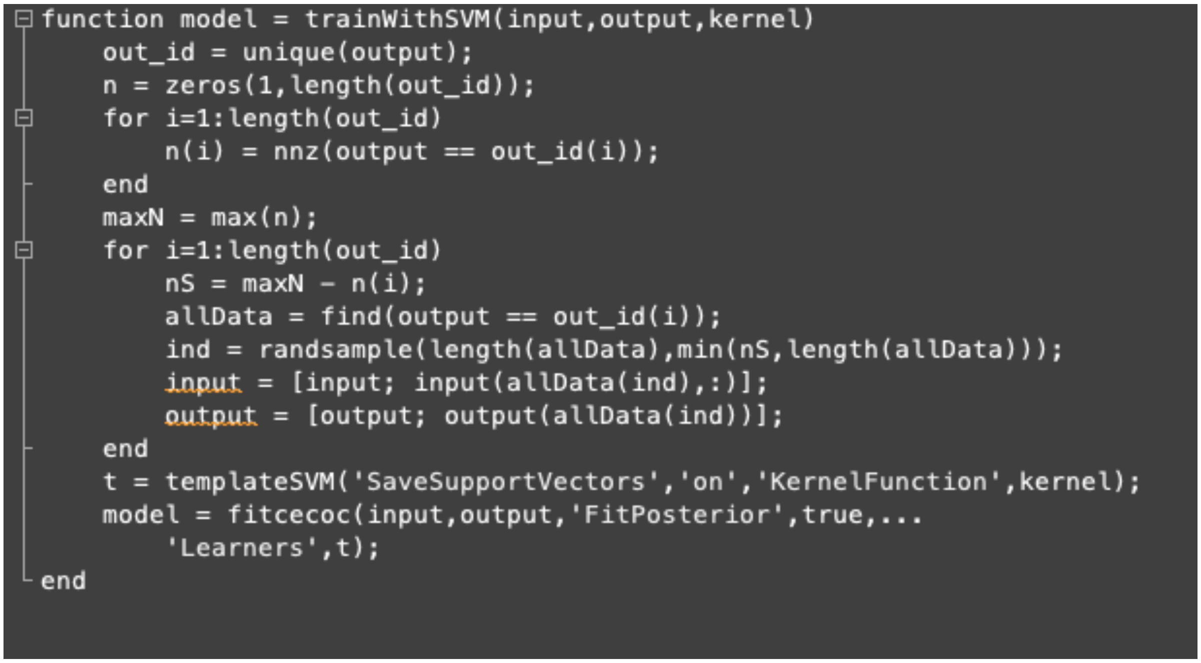
The function `templateSVM` is used to create the support vector machine template. `t=templateSVM (Name, Value)` generates a template that has possibilities of extra options specified by one of the multiple name-value pair arguments. For example, the box constraint, the kernel function, can be set. Whether there is a standardisation of the predictors or not can also be specified. All unspecified options appear empty if t, is displayed in the command window. Empty options will be replaced with the corresponding default values at the training phase. The display of t can be accessed with a code, and the results of the show are also represented in figure 55.



```
>> t = templateSVM('SaveSupportVectors',false,'KernelFunction','Linear')

t =

Fit template for classification SVM.

                     Alpha: [0×1 double]
             BoxConstraint: []
                 CacheSize: []
             CachingMethod: ''
                ClipAlphas: []
    DeltaGradientTolerance: []
                   Epsilon: []
              GapTolerance: []
              KKTTolerance: []
            IterationLimit: []
            KernelFunction: 'linear'
               KernelScale: []
              KernelOffset: []
     KernelPolynomialOrder: []
                  NumPrint: []
                        Nu: []
           OutlierFraction: []
          RemoveDuplicates: []
           ShrinkagePeriod: []
                    Solver: ''
           StandardizeData: []
        SaveSupportVectors: 0
            VerbosityLevel: []
                   Version: 2
                    Method: 'SVM'
                      Type: 'classification'

fx >> 
```

Figure 55 Display of t from the command window

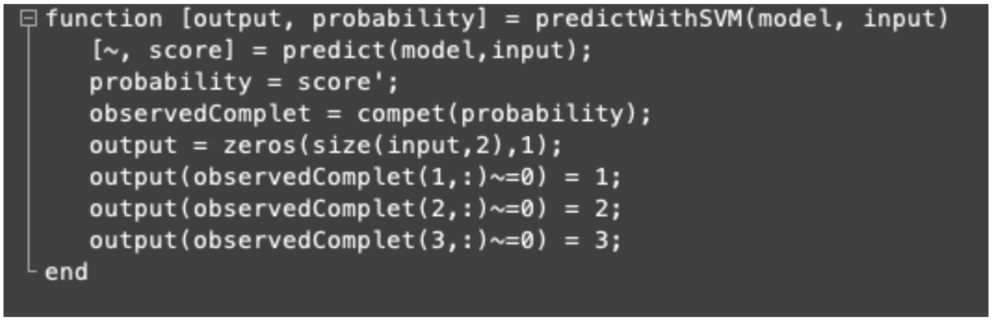
The function `trainWithSVM` is created and used to train the model.



```
function model = trainWithSVM(input,output,kernel)
    out_id = unique(output);
    n = zeros(1,length(out_id));
    for i=1:length(out_id)
        n(i) = nnz(output == out_id(i));
    end
    maxN = max(n);
    for i=1:length(out_id)
        nS = maxN - n(i);
        allData = find(output == out_id(i));
        ind = randsample(length(allData),min(nS,length(allData)));
        input = [input; input(allData(ind),:)];
        output = [output; output(allData(ind))];
    end
    t = templateSVM('SaveSupportVectors','on','KernelFunction',kernel);
    model = fitcecoc(input,output,'FitPosterior',true,...
        'Learners',t);
end
```

Figure 56 function `trainWithSVM`

The function `predictWithKNN` returns probabilities of element being assigned to a particular class and the output is in the form of numbers 1,2,3 (1 = 'i', 2 = 'M', 3 = 'o').



```
function [output, probability] = predictWithSVM(model, input)
   [~, score] = predict(model,input);
   probability = score';
   observedComplet = compet(probability);
   output = zeros(size(input,2),1);
   output(observedComplet(1,:)~=0) = 1;
   output(observedComplet(2,:)~=0) = 2;
   output(observedComplet(3,:)~=0) = 3;
end
```

Figure 57 function `predictWithKNN`

An ECOC model is created by the function `fitecoc(_, Name, Value)` (fit multiclass models for support vector machines or other classifiers). It is for multiclass SVMs. ECOC stands for Error-Correcting Output Code. Generalising SVMs to deal with multiclass problems is one of the important research activities in machine learning. The usual practice is to divide a multiclass problem into various two-class problems and then associate them to obtain a classification in multiple classes. Error-Correcting Output Code is one of the most used combination ways (Dietterich and Bakiri, 1995), called ECOC SVMs. ECOC is an ensemble method mostly used in classification problems with various classes. The idea to deal with a multi-class problem is to divide the problem into multiple smaller classification tasks called class binarisation. Two-class problems can have a solution with the use of binary classifiers. The results will be added together to solve the original multiclass problem. Various approaches in class binarisation are one-versus-one, one-versus-all, and error-correcting output codes (Dietterich and Bakiri, 1995). There is division in the multiclass problem into a series of binary problems in one-versus-all. For each class, a binary classifier is trained to make a difference between the classes patterns and the remaining classes patterns. A classifier is trained for each possible pair of classes when the one-versus-one approach is used. Voting or committee procedure is used to give a final classification prediction. When multiclass problem is divided into various binary problems, it represents the concept of the ECOC framework. A two meta-class problem is used to train each classifier. Each meta-class has a few combinations of the original classes. There are two main stages in the ECOC method: encoding and decoding. Encoding is designing the code matrix (a discrete decomposition matrix) in each problem. A codeword corresponds to a row of the codematrix. Each class is represented by multiple bits. Each bit can identify the membership of the class to a classifier (Escalera, Pujol and Radeva, 2009). For each class, a codeword of length n can be created. Each bit of the code represents the response of a given dichotomised (coded by +1, -1, according to their class set membership) (Escalera, Radeva & Pujol, 2008). The final classification decision is obtained in the decoding phase-out of the outputs of binary classifiers. When using an unlabelled test sample, each binary classifier can choose from the two metaclasses used during the training. There is a comparison of the vector to each class codeword of the matrix. There is an assignment of the test sample to the class with the closest codeword based on some distance measure. This binary strategy was suggested in various research papers and showed good results. A third symbol was added to the coding matrix by Allwein et al. (Allwein, Schapire, Singer, 2000), and it improves the representability of the ECOC technique. The elements that are part of the coding matrix is taken from {-1, 0, +1} with this new symbol. Classes with zero values are not taken into consideration for that dichotomised. Figure 58 is an illustration of various ECOC designs. (a) represents one-versus-all, (b) one-versus-one, (c) dense random, and (d) sparse random ECOC designs.

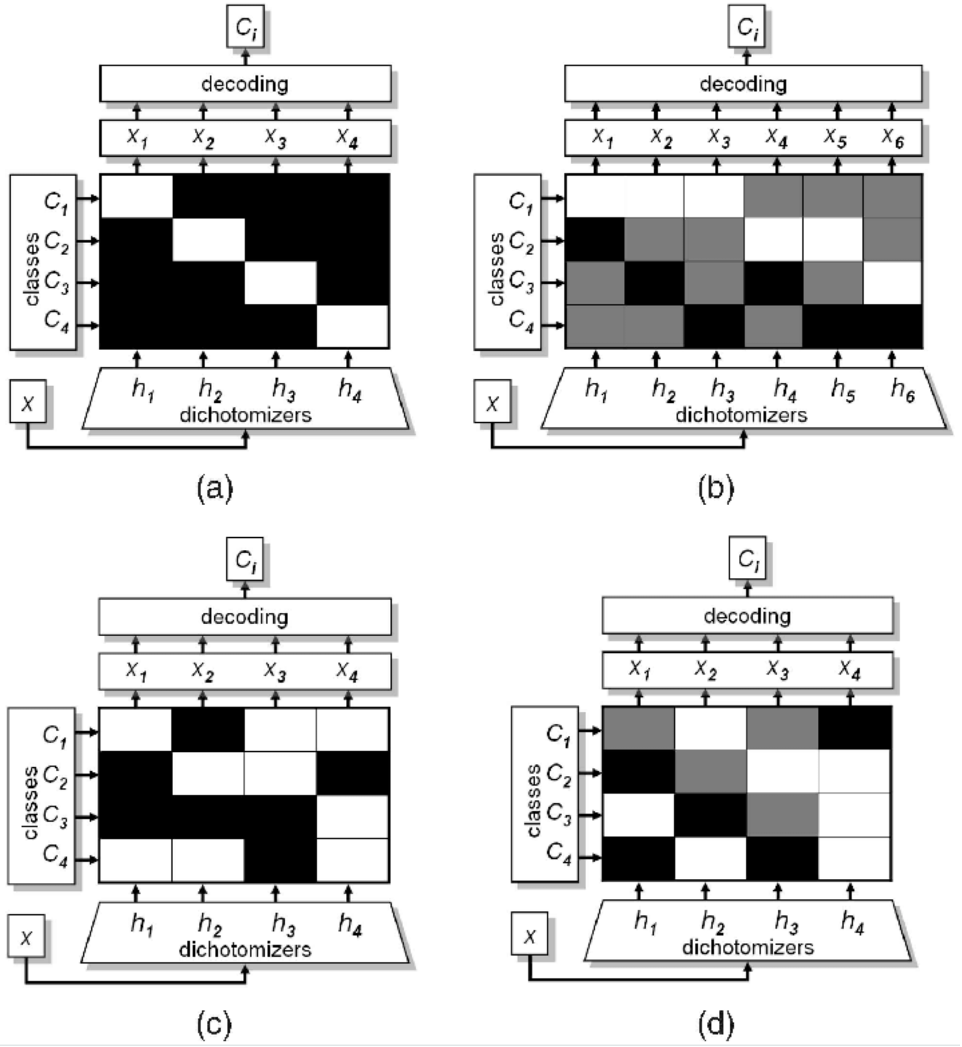
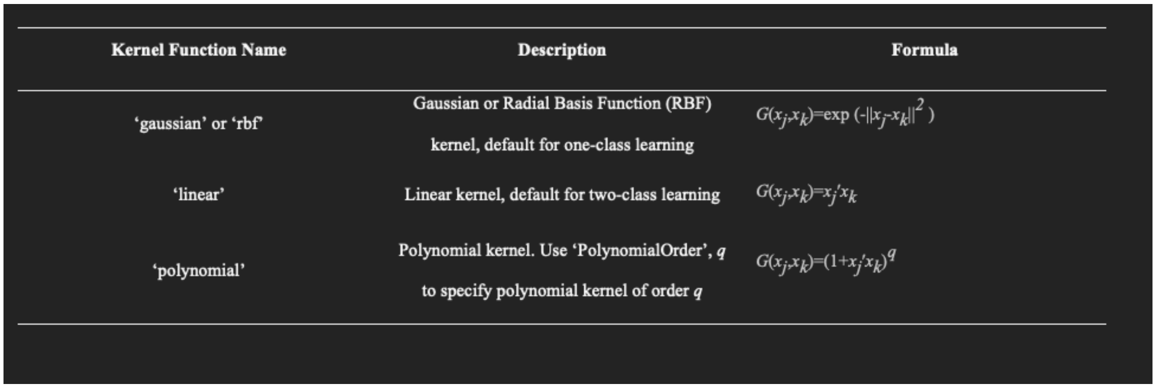


Figure 58 ECOC designs

Source: Escalera, S., Tax, D.M.J., Pujol, O., Radeva, P. and Duin, R.P.W. (2008)

+1 corresponds to the white regions. The black regions correspond to -1. Zero corresponds to the grey regions. In this example, there is the codification of four. A codeword for each class representing the rows of the coding matrix is generated. Each of the columns represents a binary problem. +1 positions correspond to the classes for the first group of a classifier. -1 positions of the column are the classes of the second group of a classifier. For this problem, there are six binary problems. A reduction of the bias and variance errors of the base classifiers are demonstrated with an ECOC ensemble (Kong and Dietterich, 1995), (Windeatt and Ghaderi, 2003). A comparison of different approaches for multiclass classification SVM problems has been described by Hsu and Lin (Hsu and Lin, 2002). The comparison includes oneversus-one, one-versus-all, and a DAG SVM. Based on their results, one-versus-one method is better than other approaches. Only ten datasets were used for this comparison. Two of those ten datasets have more than seven classes. García-Pedrajas et al. (García-Pedrajas and Ortiz-Boyer 2011) wrote and critically assessed the three basic multiclass methods. From the results, ECOC and one-versus-one have the best performances. Support vectors and the labels and the estimated α coefficients are found in '`SaveSupportVectors`' that corresponds to properties of the resulting model. They are specified as a pair consisting of '`SaveSupportVectors`' with true or false separated by a comma. The model that results when '`SaveSupportVectors`' is true will save the support vectors in the `SupportVectors` property and save their labels in the `SupportVectorsLabels` property. The estimated α coefficients are held in the Alpha property of the SVM leaners. The model that results when '`SaveSupportVectors`' is false and '`KernelFunction`' is 'Linear' does not save the support vectors or the related estimates. If you want to reduce memory consumption by compact SVM classifiers, it is necessary to have a specification of '`SaveSupportVectors`'. The Gram matrix is computed by '`KernelFunction`'. It is specified as a pair that consists of '`KernelFunction'and` '`gaussian`' or '`rbf'`, '`linear`' or '`polynomial'` and are separated by a comma. The Gram matrix also named the Gramian matrix, is a linear algebra term. With a set V of m vectors, the Gram matrix corresponds to m-by-m matrix of all possible inner products of V.A Gram matrix is formed by the algorithm using the predictor matrix columns for nonlinear SVM. The inner product of the predictors is replaced by the dual formalisation with the elements obtained from the Gram matrix also named the 'kernel trick'. `'gaussian'` or `'rbf'` is the default for one-class '`linear'` is the default for two-classes learning. Suppose $G\ (x5,\ x.)$ is element $(j,\ k)$ of the Gram matrix, where $x5$ and $x.$ are p-dimensional vectors representing observations j and k in X. Table 5 describe the supported kernel function names and their functional forms.

Table 5 Supported Kernel function names and their functional forms



Kernel Function Name	Description	Formula
'gaussian' or 'rbf'	Gaussian or Radial Basis Function (RBF) kernel, default for one-class learning	$G(x_j,x_k)$=exp $(-\|x_j-x_k\|^2)$
'linear'	Linear kernel, default for two-class learning	$G(x_j,x_k)=x_j'x_k$
'polynomial'	Polynomial kernel. Use 'PolynomialOrder', *q* to specify polynomial kernel of order *q*	$G(x_j,x_k)=(1+x_j'x_k)^q$

'`polynomial'` corresponds to the polynomial kernel. The '`PolynomialOrder`', p is equal to a polynomial kernel of order p. Transforming scores to posterior probabilities is also called '`FitPosterior`'. It is specified as a pair that consists of '`FitPosterior`' and true (1) or false (0) separated by a comma. It will transform binary-learner classification scores into posterior probabilities when '`FitPosterior`' is true. Posterior probabilities can be obtained by using predict. `Fitecoc` does not support fitting posterior probabilities when the binary learners (Learners) correspond to linear classification models implementing SVM. Logistic regression is needed to obtain posterior probabilities for linear classification models. Binary learner templates are defined as a pair that consists of 'Learners' and a character vector, cell vector of template objects or template objects. In this implementation, the 'Learners' corresponds to a template object. Each binary learner is trained following the options that are stored. The template object has been created using `templateSVM`. So far, the `trainWithSVM` function has been discussed. `PredictWithSVM` function will return probabilities of the element assigned to the class and the output in numbers 1,2,3 (1= 'i', 2='M', 3='o') based on the trained support vector machine model. The classifier uses the predict function available in MATLAB. The syntax is as follow: `[output, probability] = predictWithSVM (model, input).`

5.3.2 Hyperparameters tuning

Several SVM parameters are modifiable. 'SaveSupportVectors' and 'KernelFunction' are among those. 'SaveSupportVectors' can be 'true' or 'false' and 'KernelFunction' can be 'linear', 'rbf' or polynomial.'BoxConstraints' is another modifiable parameter. For one-class learning, the value is set to 1. An example of syntax is 'BoxConstraints',100. This parameter can supervise the maximum penalty that is imposed on margin-violating observations. It is preventing overfitting, also referred to as regularisation. Increasing the value of this parameter will make the SVM classifier have fewer support vectors, and the training is longer. Few runs have been executed with box constraints ranging from 1 to 1000. 'CacheSize'is a modifiable parameter. When 'CacheSize' is set to 'maximal', it keeps enough memory to hold the entire m-by-m Gram matrix discussed earlier. When 'CacheSize' is a positive scalar, it keeps CacheSize megabytes of memory for the classifier’s training. For large problems, it is recommended to use enough cache size. The default value is 1000. An example of the written code used is: 'CacheSize', 'maximal'.' IterationLimit'is another parameter that is modifiable. It corresponds to the maximal number of numerical optimisation iterations. 'IterationLimit'is characterised as the pair consisting of 'IterationLimit' and a positive integer. It returns a model that is trained even if the optimisation routine does not converge. Mdl.ConvergenceInfo does contain convergence details. The optimisation is slowed down when the iteration limit is very low or very high. When the iteration limit is too tight, the algorithm spends too much time doing optimisation of the dual variables of a single example 'ClipAlphas' is another parameter that can be modifiable. It is written as 'ClipAlphas' and ‘true' or 'false'. If 'false', the software is not modifying the alpha coefficients during the optimisation. 'ClipAlphas' can affect SMO and ISDA convergence. 'Solver' is another parameter that can be changed. It is specified as the comma-separated as 'Solver' and either 'ISDA' or 'L1QP' or 'SMO'. The default is 'ISDA' if 'OutlierFraction' is set to a positive value and in the case of two-class learning. It will be 'SMO' otherwise. 'SMO' refers to Sequential Minimal Optimization (Fan, Chen and.

Lin, 2005). SMO was discussed earlier. Working set selection is decisive in decomposition methods when support vector machines are trained. Fan et al. introduced a new technique for working set selection. It is for SMO-type decomposition methods. It will use a second order information to obtain fast convergence. Experimentations used by Fan et al. indicate that the method they suggested is more rapid than current selection methods that use first-order information. `'ISDA'` refers to Iterative Single Data Algorithm (Kecman, Huang and Vogt, 2005). SMO is the most popular approach for solving SVM problems. It performs a series of two-point optimisations. ISDA updates one Lagrange multiplier with each iteration. ISDA is often conducted without the bias term b with a small positive constant as the kernel function. Dropping b drops the sum constraint in the dual equation. This allows updating one Lagrange multiplier in each iteration, making it easier than SMO to remove outliers. The simulation results from Kecman et al. indicated that models produced by ISDAs, either with the bias term b or without it, are equivalent to standard SMO based algorithms concerning the generalisation performance. With the use of an appropriate k value, ISDAs will perform faster than the standard SMO algorithms when it is large scale classification problems because ISDAs are more straightforward, and there is a decrease in the number of support vectors chosen after the inclusion of an explicit bias b. The conclusion was that ISDAs are great tools when there is a need to solve large scale SVMs problems containing large training datasets. It is faster and provides the same generalisation results as the standard SMO algorithms. ISDA for two-class classification can have an extension to the multiclass problem. This approach will decrease the complexity of computation, and it requires simple iterative procedures that involve matrix addition and multiplication. There is also a guarantee of convergence of the algorithm (Kecman, Huang and Vogt, 2005). Few runs were performed with the use of the ISDA algorithm. To use the ISDA algorithm, `'OutlierFraction'` needs to be defined. '`OutlierFraction`' is the proportion of outliers in the data used for training. It is written as `'OutlierFraction'` and a numeric scalar ranging from 0 and 1.

SVMs can be affected by outliers, and methods have been developed to mitigate the effects of outliers on SVMs. Some researchers have used schemes that identify possible outliers, assigning a confidence value that indicates how likely a point is believed to be an outlier.

In implementing this, researchers have developed the ideas of fuzzy SVMs and weighted SVMs (Lin and Wang, 2004), (Lin and Wang, 2002), (Suykens, De Brabanter, Lukas and Vandewalle, 2002), (Tsujinishi and Abe, 2003), (Wang and Chiang, 2007). Lin et al. results show that the generalisation error on fuzzy SVMs is comparable to other methods on the benchmark dataset. Suykens et al. discuss LS-SVM (Least Squares SVMs).LS-SVMs is a version of an SVM that include equality and not inequality constraints. Tsujinishi et al. referred to fuzzy LS-SVMs that can resolve unclassifiable regions for multiclass problems. They indicated that, with the use of benchmark datasets, the performance of fuzzy LS-SVMs is like fuzzy SVMs.

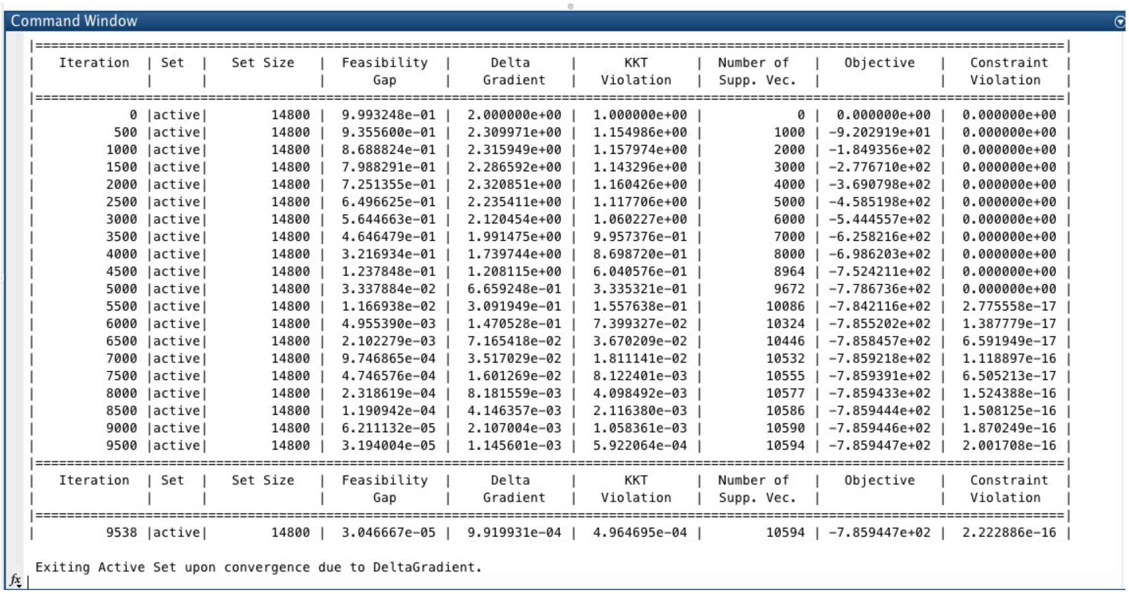
Fuzzy SVMs with the average operator did not show superior performance. Wang et al. introduced a text categorisation system to solve the multi-class categorisation problem. It was composed of two modules: The processing module and the classifying module. They concluded that the OAA-FSVM (One-Against-All Fuzzy SVM) method had outperformed OAA-SVM for the multiclass text categorisation problem. Some of these studies show only incremental improvements over the standard SVM method. Some further discussion would be to evaluate to what degree outliers affect SVM models, how much SVM models can be improved in ideal and real-world situations, or what methods can be used to deal with outliers. The last solver to discuss is the 'L1QP'. 'L1QP'is using quadprog to implement L1 soft-margin minimisation by quadratic programming. This option requires an Optimization Toolbox license. L1QP uses a generalised algorithm for QP problems. SMO is a specialised algorithm for SVMs. The optimisation toolbox was not purchased as part of the MATLAB license, and L1QP was not used during the implementation. Genetic algorithm and pattern search solvers support algorithmic customisation with the optimisation toolbox.

`'DeltaGradientTolerance'` is another parameter that can be modified. It is written as a pair consisting of `'DeltaGradientTolerance'` and a nonnegative scalar. `'DeltaGradientTolerance'` is equivalent to the tolerance for the gradient difference between upper and lower violators obtained by SMO or ISDA. If `'DeltaGradientTolerance'` is 0, then MATLAB will not use patience for the gradient difference to check the optimisation convergence. When the solver is SMO, the defaults are 1e-3. When the solver is ISDA, the default is 0. Tolerance specifies the maximum gradient of the quadratic function used to compute support vectors. Training is terminated when the rise of the optimises function is less than or equal to the tolerance value. Typically, there is no need to change the default value. `'GapTolerance'` is another parameter that can be modified. It is written as a pair consisting of `'GapTolerance'` with a nonnegative scalar. It is equivalent to the feasibility gap tolerance that SMO or ISDA obtains. When the value is equal to 0, then MATLAB does not use the feasibility gap tolerance when checking for optimisation convergence. '`KernelOffset`' is another parameter that can be modified. It is written as a pair consisting of `'KernelOffset'` with a nonnegative scalar.thoseMATLAB will add `'KernelOffset' to` each element of the Gram matrix. If the solver is SMO, the default value is 0. It's 0.1 if the solver is ISDA. The offset parameter is sometimes called 'bias' in classification tasks, and its intuitive understanding does not have to do with what kind of kernel is used. It is used to compensate for feature vectors not centred around 0. For example, you can have some feature vector x whose parameters are always negative. The set of weights w used in the SVM (for instance linear), will perhaps transform the features into the range [0,1] so they will always be negative. Those elements that belong to class 1 fall in the range [0,0.5], and those from class 2 fall into the range [0.5,1]. The SVM uses 0 as the threshold breakpoint to classify into a class. If greater than 0, it is an element of class 1 and if less than 0, it is an element of class -1. But in this case, all the elements will be classified into class 2. However, with a bias of 0.5 (in the linear case), they will be classified correctly. This geometric interpretation does not work for more complex kernels, but this idea is the same: the bias term attempts to compensate for features not centred around 0. If the features are centred around zero, the bias term is not always needed. '`KernelScale`' is another parameter that can be modified. Gamma (γ), also named `KernelScale` in MATLAB, controls overfitting in SVM. It is written as a pair consisting of `'KernelScale'` with a positive scalar or 'auto'. MATLAB will divide all elements of the predictor matrix X by the value of'`KernelScale`'. If the solver is SMO, it will apply the appropriate kernel norm to compute the Gram matrix. If it is written 'auto', MATLAB will select a relevant scale factor using a heuristic procedure.; The heuristic procedure will be using subsampling; therefore, estimates can vary from one call to another. To obtain the same results, setting a random number using rng before the training is mandatory. `rng` corresponds to control random number generation. '`KKTTolerance`' is another parameter that can be modified. It is written as a pair consisting of `'KKTTolerance'` and a positive scalar. `KKTTolerance` is equivalent to Karush-KuhnTucker complementary conditions violation tolerance. In SVM, the KKT complementary conditions are:

$$\begin{gathered} \alpha_j[y_j f(x_j) - 1 + \xi_j] = 0 \\ \xi_j(C - \alpha_j) = 0 \end{gathered} \tag{5.3.1}$$

If `'KKTTolerance'` equals 0, then MATLAB will not use the KKT complementary violation tolerance to check for optimisation convergence. If the solver is SMO, the default value is 0; otherwise, 1e-3 is the solver is ISDA. Karush-Kuhn-Tucker complementary conditions have an essential role in the use of constrained optimisation. KKT conditions come from Harold W. Kuhn and Albert W. Tucker.

The first published the conditions in 1951 (Kuhn and Tucker, 1951). 'Nu' is another parameter that can be modified. It is written as a pair consisting of 'Nu' with a positive scalar. 'Nu' is between 0 and 1. It corresponds to the v parameter for one-class learning. Setting. It is used for one-class learning, but the implementation is a multi-class learning problem. The value was changed to 0.25. No changes in the accuracy were observed. '`Numprint`' is another parameter that can be modified. It is written as a pair consisting of `'Numprint'` with a nonnegative integer. It corresponds to the number of iterations between optimisation diagnostic message output. If ‘Verbose',1 and '`Numprint`', `Numprint` are used, then MATLAB will display all optimisation diagnostic messages from SMO and ISDA every `Numprint` iteration in the command window. 'Verbose' is another parameter that can be modified. It is written as a pair consisting of `'Verbose'` with either 0,1, or 2. It corresponds to the verbosity level. It controls the amount of optimisation information that MATLAB will display in the command window and save as the structure `Mdl.ConvergenceInfo`. History. MATLAB will not display or save convergence information if the value is set at 0 (default). MATLAB will display diagnostic messages and save convergence criteria for every `Numprint` iteration if the value is 1. Figure 59 represents the diagnostic message and convergence criteria displayed in the command window during the implementation with 'Numprint',500 and 'Verbose',1.



```
|==========================================================================================================================|
|  Iteration  | Set  |  Set Size  | Feasibility  |    Delta     |     KKT      | Number of  |   Objective   |  Constraint  |
|             |      |            |     Gap      |   Gradient   |  Violation   | Supp. Vec. |               |  Violation   |
|==========================================================================================================================|
|           0 |active|      14800 | 9.993248e-01 | 2.000000e+00 | 1.000000e+00 |          0 |  0.000000e+00 | 0.000000e+00 |
|         500 |active|      14800 | 9.355600e-01 | 2.309971e+00 | 1.154986e+00 |       1000 | -9.202919e+01 | 0.000000e+00 |
|        1000 |active|      14800 | 8.688824e-01 | 2.315949e+00 | 1.157974e+00 |       2000 | -1.849356e+02 | 0.000000e+00 |
|        1500 |active|      14800 | 7.988291e-01 | 2.286592e+00 | 1.143296e+00 |       3000 | -2.776710e+02 | 0.000000e+00 |
|        2000 |active|      14800 | 7.251355e-01 | 2.320851e+00 | 1.160426e+00 |       4000 | -3.690798e+02 | 0.000000e+00 |
|        2500 |active|      14800 | 6.496625e-01 | 2.235411e+00 | 1.117706e+00 |       5000 | -4.585198e+02 | 0.000000e+00 |
|        3000 |active|      14800 | 5.644663e-01 | 2.120454e+00 | 1.060227e+00 |       6000 | -5.444557e+02 | 0.000000e+00 |
|        3500 |active|      14800 | 4.646479e-01 | 1.991475e+00 | 9.957376e-01 |       7000 | -6.258216e+02 | 0.000000e+00 |
|        4000 |active|      14800 | 3.216934e-01 | 1.739744e+00 | 8.698720e-01 |       8000 | -6.986203e+02 | 0.000000e+00 |
|        4500 |active|      14800 | 1.237848e-01 | 1.208115e+00 | 6.040576e-01 |       8964 | -7.524211e+02 | 0.000000e+00 |
|        5000 |active|      14800 | 3.337884e-02 | 6.659248e-01 | 3.335321e-01 |       9672 | -7.786736e+02 | 0.000000e+00 |
|        5500 |active|      14800 | 1.166938e-02 | 3.091949e-01 | 1.557638e-01 |      10086 | -7.842116e+02 | 2.775558e-17 |
|        6000 |active|      14800 | 4.955390e-03 | 1.470528e-01 | 7.399327e-02 |      10324 | -7.855202e+02 | 1.387779e-17 |
|        6500 |active|      14800 | 2.102279e-03 | 7.165418e-02 | 3.670209e-02 |      10446 | -7.858457e+02 | 6.591949e-17 |
|        7000 |active|      14800 | 9.746865e-04 | 3.517029e-02 | 1.811141e-02 |      10532 | -7.859218e+02 | 1.118897e-16 |
|        7500 |active|      14800 | 4.746576e-04 | 1.601269e-02 | 8.122401e-03 |      10555 | -7.859391e+02 | 6.505213e-17 |
|        8000 |active|      14800 | 2.318619e-04 | 8.181559e-03 | 4.098492e-03 |      10577 | -7.859433e+02 | 1.524388e-16 |
|        8500 |active|      14800 | 1.190942e-04 | 4.146357e-03 | 2.116380e-03 |      10586 | -7.859444e+02 | 1.508125e-16 |
|        9000 |active|      14800 | 6.211132e-05 | 2.107004e-03 | 1.058361e-03 |      10590 | -7.859446e+02 | 1.870249e-16 |
|        9500 |active|      14800 | 3.194004e-05 | 1.145601e-03 | 5.922064e-04 |      10594 | -7.859447e+02 | 2.001708e-16 |
|==========================================================================================================================|
|  Iteration  | Set  |  Set Size  | Feasibility  |    Delta     |     KKT      | Number of  |   Objective   |  Constraint  |
|             |      |            |     Gap      |   Gradient   |  Violation   | Supp. Vec. |               |  Violation   |
|==========================================================================================================================|
|        9538 |active|      14800 | 3.046667e-05 | 9.919931e-04 | 4.964695e-04 |      10594 | -7.859447e+02 | 2.222886e-16 |

 Exiting Active Set upon convergence due to DeltaGradient.
```

Figure 59 Diagnostic message and convergence criteria

MATLAB will display diagnostic messages and save convergence criteria at every iteration if the value is set a 2. '`PolynomialOrder`' is another parameter that can be modified. It is written as a pair consisting of `'PolynomialOrder'` and a positive integer. It corresponds to the polynomial kernel function order. The default value is 3. For this implementation, various kernel functions have been used. The most efficient was the polynomial kernel with polynomial order 3. It corresponds to the literature findings related to the efficiency of kernel functions in multi-class SVM (Rajendran & Kalpana, 2011). The polynomial kernel is often used with SVMs or kernelised models. It represents the similarity of vectors (training sample) in a feature space over polynomials of the original variables, allowing learning of non-linear models. The polynomial kernel looks not exclusively at the given features of input samples to determine their similarity, but it also looks at combinations of these. For regression analysis, the combinations refer to interaction features. The feature space of a polynomial kernel equals that of polynomial regression but without the combinatorial blow-up in the number of parameters to be learned. The features correspond to logical conjunctions of input features when the input features are binary-valued (Booleans) (Goldberg and Elhadad, 2008).

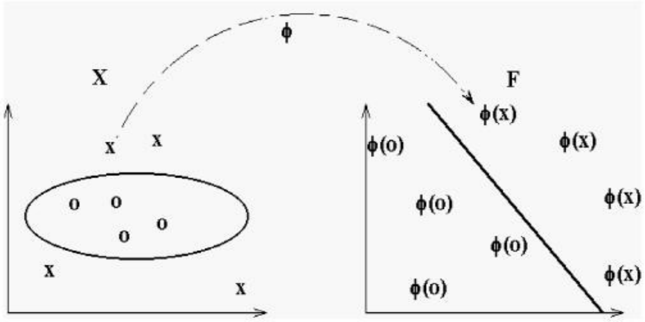


Figure 60 Illustration of the mapping φ

Source: Wikipedia Contributors (2019)

Figure 60 represents a set of samples in the input space on the left. On the right, it means the same sample in the feature space where the polynomial Kernel $K\ (x, y)$ (for some values of the parameters c and d) is the inner product. The hyperplane learned in feature space by an SVM is an ellipse in the input space. For degree-d polynomials, the polynomial kernel is defined as:

$$K(x, y) = (x^4 y + c)^{\%} \qquad (5.3.2)$$

where x and y are vectors in the input space, i.e., vectors of features computed from training or test samples and c ≥ 0 is a free parameter trading off the influence of higher-order versus lower order terms in the polynomial. As a kernel, K corresponds to an inner product in a feature space based on some mapping Figure 60 represents a set of samples in the input space on the left. On the right, it represents the same sample in the feature space where the polynomial Kernel $K(x, y)$ (for some values of the parameters j).

$$K(x, y) = \langle \varphi(x), \varphi(y) \rangle \qquad (5.3.3)$$

The nature of φ can be seen from an example. Let d=2, so it corresponds to the special case of the quadratic kernel. After using the multinomial theorem and regrouping$K(x, y) =$

$$(\textstyle\sum_{i=1}^{n} x_i y_i + c)^2 = \sum_{i=1}^{n}(x_i^2)(y_i^2) + \sum_{i=2}^{n}\sum_{j=1}^{i-1}(\sqrt{2x_i}\, x_j)(\sqrt{2y_i} y_j) + \sum_{i=1}^{n}(\sqrt{2cx_i})(2cy_i) + c^2 \qquad (5.3.4)$$

From this, it follows that the feature map is given by:

$$\varphi(x) = \left\langle \begin{array}{c} x_n^2, \dots, x_1^2, \sqrt{2x_n}x_{n-1}, \dots, \sqrt{2x_n x_1}, \\ \sqrt{2x_{n-1}x_{n-2}}, \dots, \\ \sqrt{2x_{n-1}x_1}, \dots, \sqrt{2x_2 x_1}, \sqrt{2cx_n}, \dots, \sqrt{2cx_1}, c \end{array} \right\rangle \qquad (5.3.5)$$

The order of polynomial in mathematics refers also to the degree of a polynomial, that is, the largest exponent (for a univariate polynomial) or sum of exponents (for a multivariate polynomial) in any of its monomials (Dos Santos and Gomes, 2002).

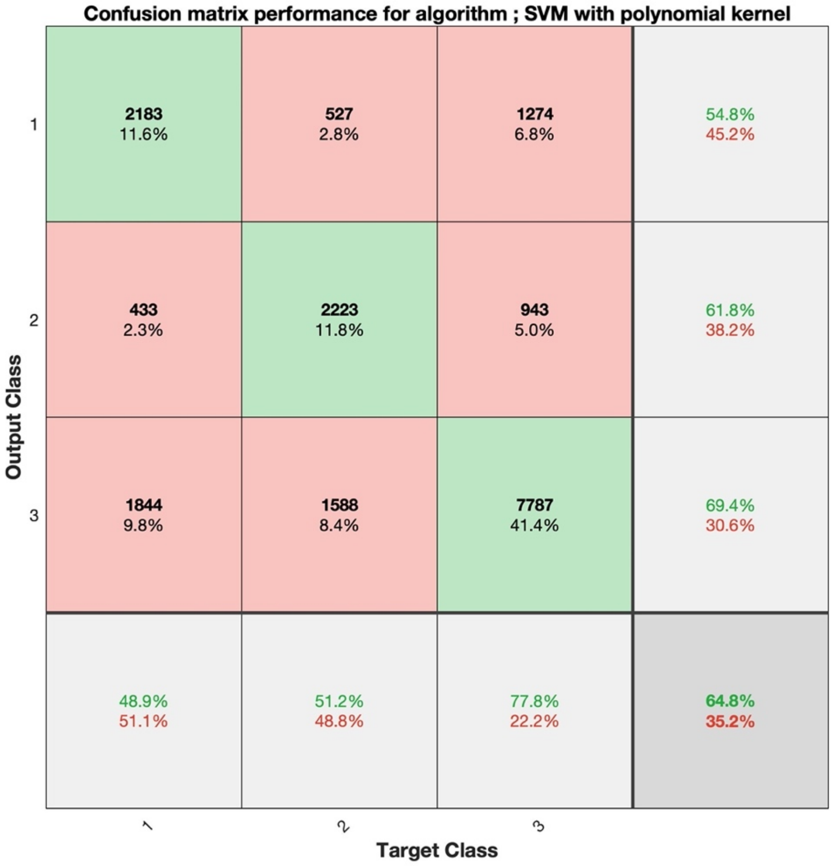
The following names are given to polynomials according to their degree. Degree 0 corresponds to non-zero constant, degree 1 is linear, degree 2 is quadratic, degree 3 is cubic, degree 4 is quartic and so on. In general, SVM works well on small datasets. A recent study evaluated the performance of SVMs with linear, quadratic, and cubic kernels in the problem of recognising 3D objects from 2D views. The paper indicates that the degree of the polynomial order plays a minor role in the results.

'`ShrinkagePeriod`' is another parameter that can be modified. It is written as a pair consisting of `'ShrinkagePeriod'` with a nonnegative integer. It corresponds to the number of iterations between the movement of observations from active to inactive set. MATLAB will not shrink the active set if it has a value of 0. Convergence can be speeded up with shrinking when the support vector set is much smaller than the amount of data in the training dataset. 1000 is a suggested starting point. 'Standardize' is another parameter that can be modified. It is written as a pair consisting of `'Standardize'` and `'true'` or 'false'. It corresponds to a flag to standardise the predictor data. MATLAB will centre and scale each column of the predictor data (x) by the weighted column mean and standard deviation if set as 'true'. The software will not standardise the data contained in the dummy variable columns and generated for categorical predictors. MATLAB will train the classifier using the standardised predictor matrix if it is set as 'true'., The unstandardised data will be stored in the classifier property x. For the function `fitecoc`, `'Cost'` is another parameter that can be modified. It is written as a pair consisting of `'Cost'` and a square matrix or structure. It corresponds to the misclassification cost. All runs executed for SVM classifiers can be found in Appendix C.

5.3.3 Performance evaluation

5.3.3.1 Confusion matrix

The confusion matrix is represented in figure 61. The diagonal cells show the number of residue positions correctly classified for each topology class. The off-diagonal cells show the number of residue positions that were misclassified. The diagonal cells correspond to observations that are correctly classified. The number of observations and the percentage of the total number of observations is shown in each cell. The column on the far right of the plot shows the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. These metrics are often called the precision (or positive predictive value) and false discovery rate. The row at the bottom of the plot shows the percentages of all the examples belonging to each class that are correctly and incorrectly classified. These metrics are called the recall (or true positive rate) and false-negative rate. The cell in the bottom right of the plot shows the overall accuracy. The best overall accuracy is 64.8%.



Confusion matrix performance for algorithm ; SVM with polynomial kernel

Output Class \ Target Class	1	2	3	
1	2183 11.6%	527 2.8%	1274 6.8%	54.8% 45.2%
2	433 2.3%	2223 11.8%	943 5.0%	61.8% 38.2%
3	1844 9.8%	1588 8.4%	7787 41.4%	69.4% 30.6%
	48.9% 51.1%	51.2% 48.8%	77.8% 22.2%	64.8% 35.2%

Figure 61 Confusion matrix for SVM classifier

5.3.3.2 Receiver operating characteristic curve

The Receiver Operating Characteristic (ROC) curve can also be looked at and is represented in figure 62. It is a plot of the true positive rate (sensitivity) versus the false positive rate (1 – specificity). True positive rate is also known as sensitivity in biomedical engineering. Sensitivity corresponds to the proportion of actual positives that are correctly identified. False-positive corresponds to fall-out. Fall-out is related to the specificity and equals to 1 - specificity. The ROC curve represents the sensitivity as a function of the fall-out. A perfect predictor would be described at 100% sensitivity. When the ROC curve is close to the upper left corner (100% sensitivity, 100% specificity), the overall accuracy is better.

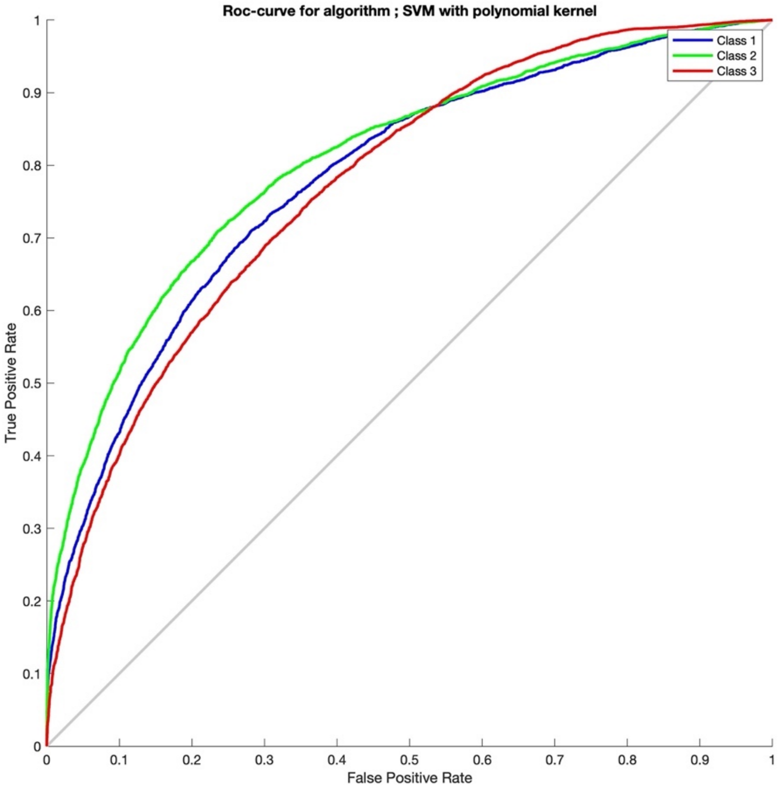


Figure 62 ROC curve for SVM classifier

5.3.3.3 Prediction quality indices

The topology prediction is evaluated in detail by calculating prediction quality indices represented in figure 63. They indicate how well a particular state is predicted and whether overprediction or underprediction has occurred. The index `pcObs(S)` was defined for state S (S = {I, O, M}) as the number of residues correctly predicted in state S, divided by the number of residues observed in state S. Similarly, the index `pcPred(S)` for state S was defined as the number of residues correctly predicted in state S, divided by the number of residues predicted in state S. These quality indices are useful for the interpretation of the prediction accuracy.

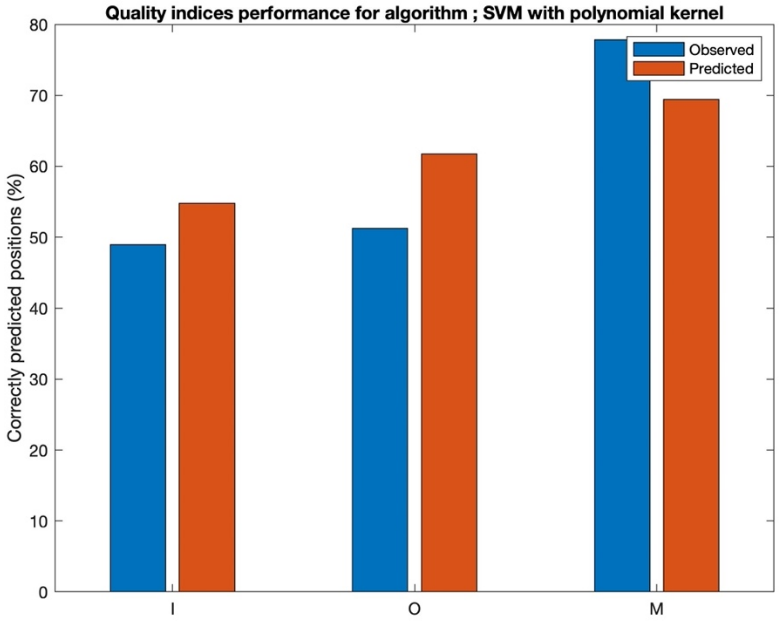


Figure 63 Quality indices of prediction for SVM classifier

5.4 Comparison of results

The best overall accuracy for ANN is 64.5%, including a TMB topology prediction of 69%. The best overall accuracy for KNN is 71.8%, including a TMB topology prediction of 72.3%. The best overall accuracy for SVM is 64.8%, including a TMB topology prediction of 69.4%. A general difference between KNN and SVM is that SVM cares for outliers better than KNN. Neural networks need large training data compared to KNN to achieve sufficient accuracy. Also, neural networks need many hyperparameters tuning compared to KNN. KNN is an easy and simple machine learning model, and there are few parameters to tune. The value k needs to be wisely selected. A value of 8 provided the best results in the implementations. For ANN, the training function had an essential impact on results and the scaled conjugate gradient backpropagation provided the best results. SVM provided better accuracy than ANN when a polynomial kernel was used.

5.5 Summary

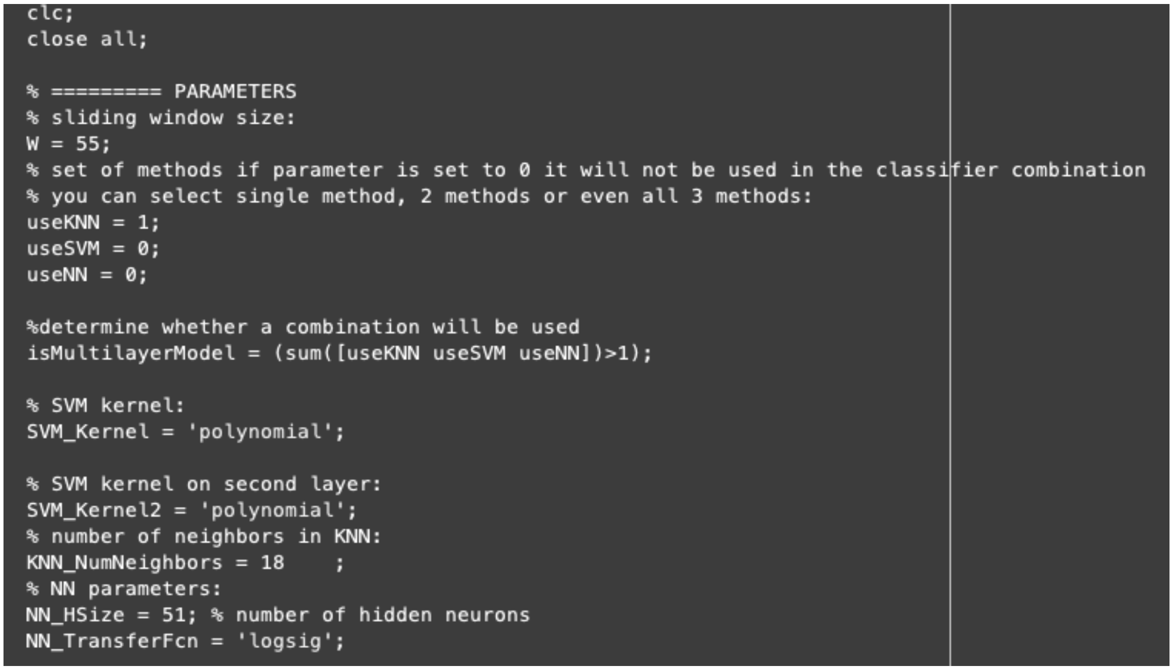
This chapter presented the implementations of an ANN, KNN and SVM in MATLAB using the Boctopus2 dataset and TopBP dataset. Models were built and trained in MATLAB. For all models, hyperparameters have been tuned, and the performance of each model was evaluated with the use of confusion matrices, ROC curves and bar charts. To maximise the performance and confirm the precision, fine-tuning the hyper-parameters was necessary. The problem of TMB topology was looked at as a pattern recognition problem as the neural network was trained to recognise the topology of the central residue most likely to occur when specific residues in the given window are observed. At each training cycle, the training sequences were presented to the network through the sliding window one residue at a time. For the KNN implementation, the classifier used the `fitcknn` function. `Fitcknn` created a model, and the tie-breaking algorithm, distance metric and number of neighbors were specified and modified. For the SVM implementation, the function `templateSVM` was used to create the support vector machine. Multiple parameters were changed, such as the box constraint or the kernel function. To analyse the model's responses, the confusion matrix was examined by considering the outputs of the models and comparing them to the expected results. The ROC curve was also used, and it corresponds to a plot of the true positive rate (sensitivity) versus the false positive rate (1-specificity). The topology prediction accuracy was finally evaluated by calculating prediction quality indices. They indicate how well a particular topology is predicted and whether overprediction or underprediction has occurred.

6. Implementation of cascading classifier

This chapter presents the implementation of the cascading classifier in MATLAB.

6.1 Model building and training

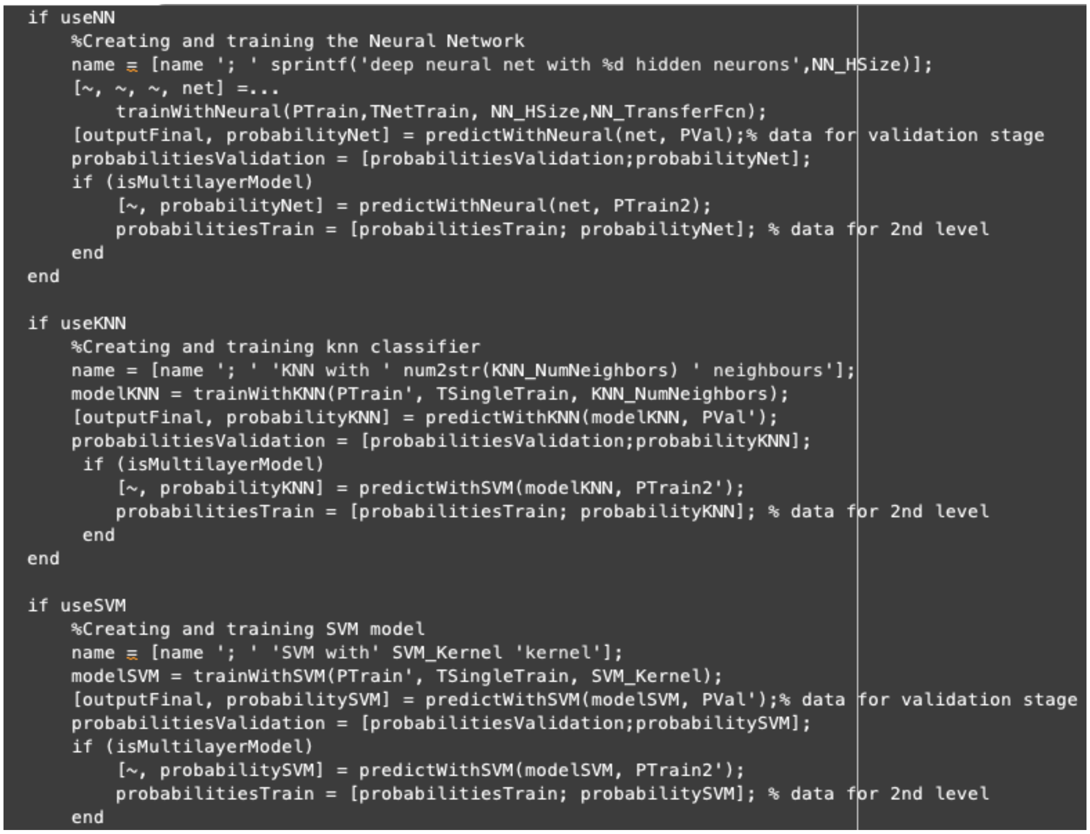
The model consists of two levels. Several selected models will be trained at the first level. The chosen algorithms (KNN, SVM or ANN) are trained to predict the values of the class as it was before. In case two or more classifiers are selected, a second level SVM classifier is trained to anticipate the value of the class based on the probability predicted by those two or more models at the first level. This is a cascading classifier as the output of the first layer corresponds to the input of the second layer. If only one model is initially selected at the first level, the second layer classifier will not be trained. The main function used for the cascading classifier implementation in MATLAB is named UseClassifiers.m. Part of the code used for the implementation in MATLAB is defined in figure 64.



```
clc;
close all;

% ========= PARAMETERS
% sliding window size:
W = 55;
% set of methods if parameter is set to 0 it will not be used in the classifier combination
% you can select single method, 2 methods or even all 3 methods:
useKNN = 1;
useSVM = 0;
useNN = 0;

%determine whether a combination will be used
isMultilayerModel = (sum([useKNN useSVM useNN])>1);

% SVM kernel:
SVM_Kernel = 'polynomial';

% SVM kernel on second layer:
SVM_Kernel2 = 'polynomial';
% number of neighbors in KNN:
KNN_NumNeighbors = 18    ;
% NN parameters:
NN_HSize = 51; % number of hidden neurons
NN_TransferFcn = 'logsig';
```

Figure 64 Part of UseClassifiers.m for cascading classifier

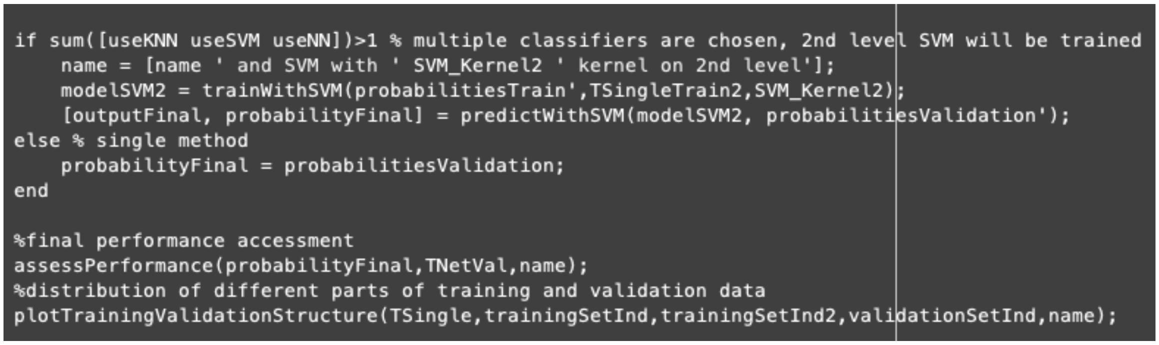
The model allows the use of a single method, two or even three methods. `useKNN =1` or `0`, `useSVM = 1` or `0` and `useNN = 1` or `0` are the list of parameters. If a parameter is equal to 0, it will not be used in the cascading classifier. All combinations are possible except for combinations when they are equal to 0. If only they are set to 1, the application will work as one classifier. The selected classifier (equals 1) will be trained and show the results. If at least two parameters are set to 1, several classifiers will be trained and then combined by one more probability classifier. The code used for creating and training the neural network at the first level is represented in figure 65.



```
if useNN
    %Creating and training the Neural Network
    name = [name '; ' sprintf('deep neural net with %d hidden neurons',NN_HSize)];
    [~, ~, ~, net] =...
        trainWithNeural(PTrain,TNetTrain, NN_HSize,NN_TransferFcn);
    [outputFinal, probabilityNet] = predictWithNeural(net, PVal);% data for validation stage
    probabilitiesValidation = [probabilitiesValidation;probabilityNet];
    if (isMultilayerModel)
        [~, probabilityNet] = predictWithNeural(net, PTrain2);
        probabilitiesTrain = [probabilitiesTrain; probabilityNet]; % data for 2nd level
    end
end

if useKNN
    %Creating and training knn classifier
    name = [name '; ' 'KNN with ' num2str(KNN_NumNeighbors) ' neighbours'];
    modelKNN = trainWithKNN(PTrain', TSingleTrain, KNN_NumNeighbors);
    [outputFinal, probabilityKNN] = predictWithKNN(modelKNN, PVal');
    probabilitiesValidation = [probabilitiesValidation;probabilityKNN];
     if (isMultilayerModel)
        [~, probabilityKNN] = predictWithSVM(modelKNN, PTrain2');
        probabilitiesTrain = [probabilitiesTrain; probabilityKNN]; % data for 2nd level
     end
end

if useSVM
    %Creating and training SVM model
    name = [name '; ' 'SVM with' SVM_Kernel 'kernel'];
    modelSVM = trainWithSVM(PTrain', TSingleTrain, SVM_Kernel);
    [outputFinal, probabilitySVM] = predictWithSVM(modelSVM, PVal');% data for validation stage
    probabilitiesValidation = [probabilitiesValidation;probabilitySVM];
    if (isMultilayerModel)
        [~, probabilitySVM] = predictWithSVM(modelSVM, PTrain2');
        probabilitiesTrain = [probabilitiesTrain; probabilitySVM]; % data for 2nd level
    end
```

Figure 65 Useclassifiers.m

When multiple classifiers are chosen, a second level SVM will be trained. The code used for that and the final performance assessment is represented below.



```
if sum([useKNN useSVM useNN])>1 % multiple classifiers are chosen, 2nd level SVM will be trained
    name = [name ' and SVM with ' SVM_Kernel2 ' kernel on 2nd level'];
    modelSVM2 = trainWithSVM(probabilitiesTrain',TSingleTrain2,SVM_Kernel2);
    [outputFinal, probabilityFinal] = predictWithSVM(modelSVM2, probabilitiesValidation');
else % single method
    probabilityFinal = probabilitiesValidation;
end

%final performance accessment
assessPerformance(probabilityFinal,TNetVal,name);
%distribution of different parts of training and validation data
plotTrainingValidationStructure(TSingle,trainingSetInd,trainingSetInd2,validationSetInd,name);
```

Figure 66 Creating and training the SVM classifier at layer 2

6.2 Hyperparameters tuning

The model has four different basic configurations:

- Configuration 1: Layer 1(KNN=1, SVM=1, ANN=1)/Layer 2 (SVM)
- Configuration 2: Layer 1(KNN=0, SVM=1, ANN=1)/Layer 2 (SVM)
- Configuration 3: Layer 1(KNN=1, SVM=0, ANN=1)/Layer 2 (SVM)
- Configuration 4: Layer 1(KNN=1, SVM=1, ANN=0)/Layer 2 (SVM)

The model allows the use of a single method, two or even three methods. KNN =1 or 0, SVM = 1 or 0, and ANN = 1 or 0 are the list of parameters. If a parameter is equal to 0, it will not be used in the cascading classifier. All combinations are possible except for combinations when they are equal to 0. If only one of them is set to 1, the application will work as one classifier. The selected classifier (equals 1) will be trained and show the results. If at least two parameters are set to 1, several classifiers will be trained and then combined by one more probability classifier. When multiple classifiers are chosen, a second level SVM will be trained. The parameters of each machine learning algorithm can be modified. Appendix D summarises all the runs executed.

6.2.1 Performance evaluation

The best results are obtained when KNN, SVM and ANN are used at layer 1 and SVM at layer 2. Multiple runs have been executed in MATLAB using different parameters configurations. Various ratio combinations such as 50:50, 60:40, 70:30, 75:25 and 80:20 have been used to split training and testing data in tsPart1. Best results for the cascading classifier were obtained using a split with 80% of data used for training and 20% used for testing in tsPart1 and 42% in tsPart2. Best results are obtained with parameters configured to a window size of 65, bits encodings of 50, Hidden layer size of 50, `logsig` transfer function, scaled conjugate gradient for the training function, `'Sum'` Performance Function and '`Dividerand`' data division for the ANN part of the cascading classifier and k-value of 8, exhaustive nearest neighbors search method, random tie-breaking algorithm for the KNN part of the cascading classifier. For the SVM part of the cascading classifier, a polynomial kernel function, '`SaveSupportVectors`' is equivalent to 'true' have been used as well as default values for the box constraint, cache size, Solver, tolerance to gradient difference, feasibility gap tolerance, Maximal number of optimisation iterations, kernel offset parameter, kernel scale, Karush-Kuhn-Tucker complementarity conditions violation tolerance, v parameter for one-class learning, number of iterations between optimisation diagnostic message output, expected proportion of outliers in training data, Polynomial kernel function order, number of iterations between the movement of observations from active to inactive set, flag to standardise predictor data and verbosity level. It is interesting to note that an increase in the amount of data does produce better results. Two runs were executed using the same parameters with different datasets available: TopBP Dataset (1x123) and BOCTOPUS2 Dataset (1x42).

The best overall accuracy obtained is 76.3%, including a TMB topology prediction of 83.1%. The accuracy of 83.1% is for one scenario combination where layer one includes SVM, KNN and ANN, and layer two include SVM when using the TopBP Dataset. For the ANN, during the implementation, the training process stops when the maximum validation failures are above six, or the maximum number of allowed iterations is reached (1000). Figure 67 represents the variation in gradient coefficient concerning the number of epochs at and the final value of the gradient at epoch 138.

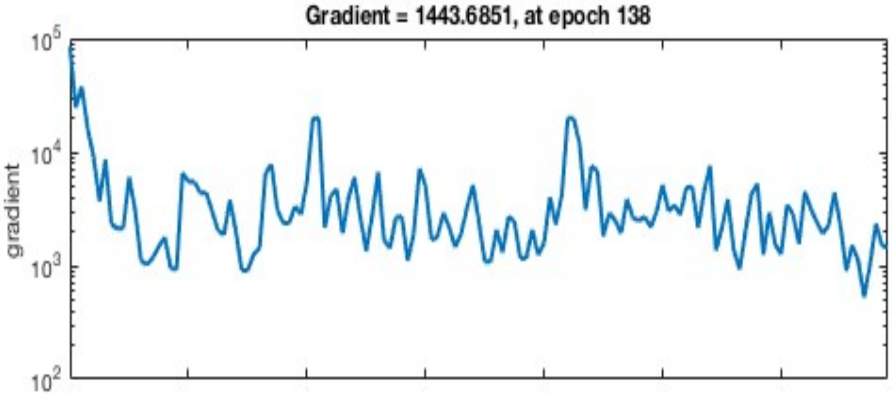


Figure 67 Magnitude of gradient

The parameter for the minimum gradient `min_grad` was set to 1e-06 and was not reached during the implementation.

For this project, the validation checks triggered the training to stop. Figure 68 represents the validation checks. MATLAB automatically stopped the training after six fails in a row.

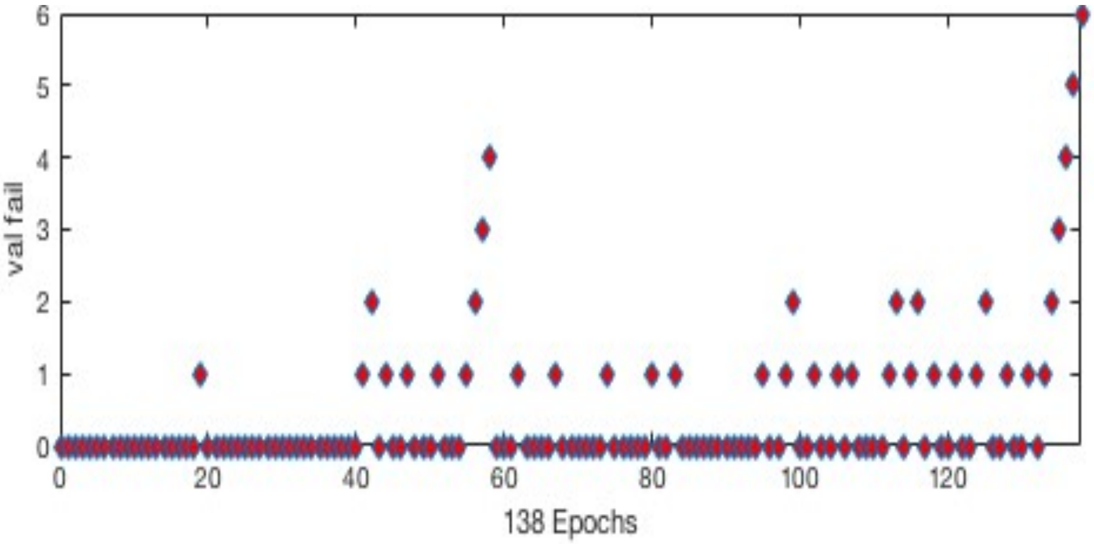


Figure 68 Validation checks

The data division is an automatic process that happens when the network is trained. During the implementation, the composition of the residues in three subsets is comparable, as seen in figures 69,70 and 71.

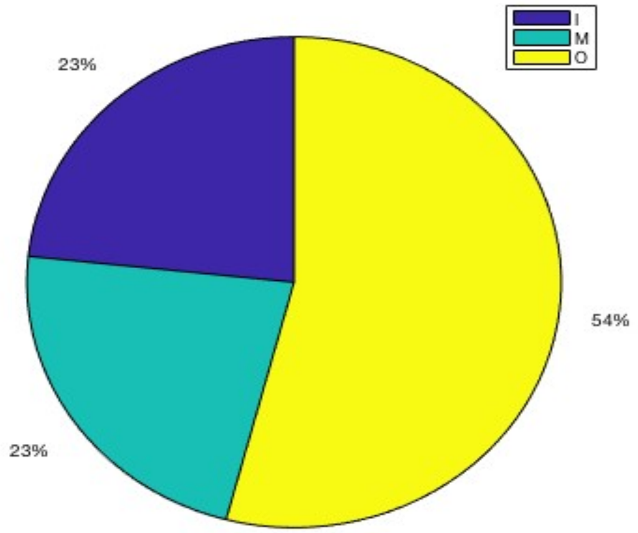


Figure 69 Topology assignments in the training data set

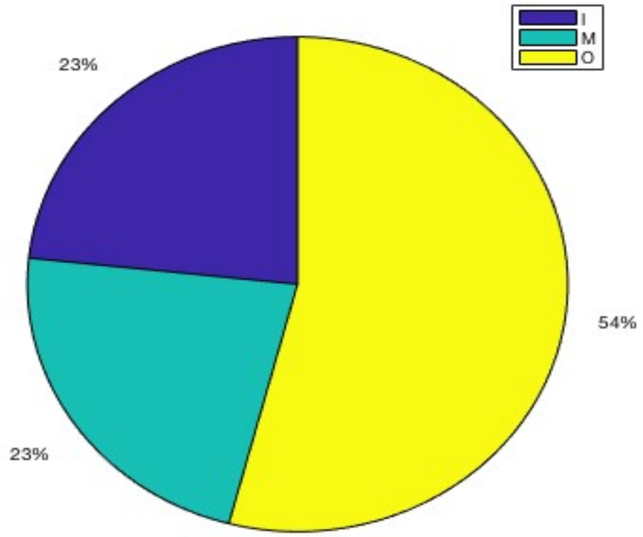


Figure 70 Topology assignments in the validation data set

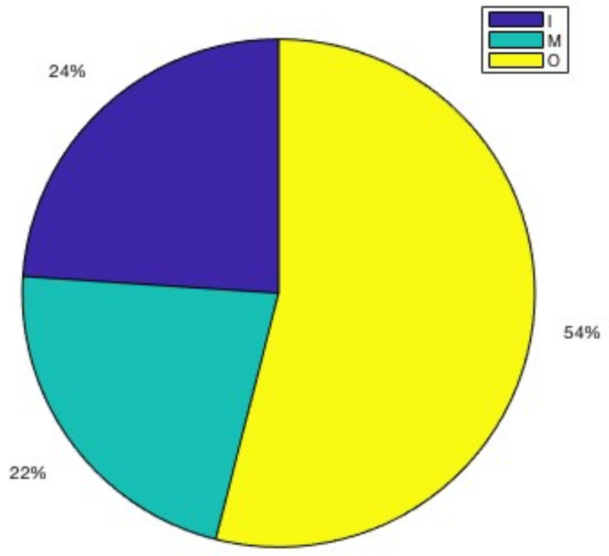
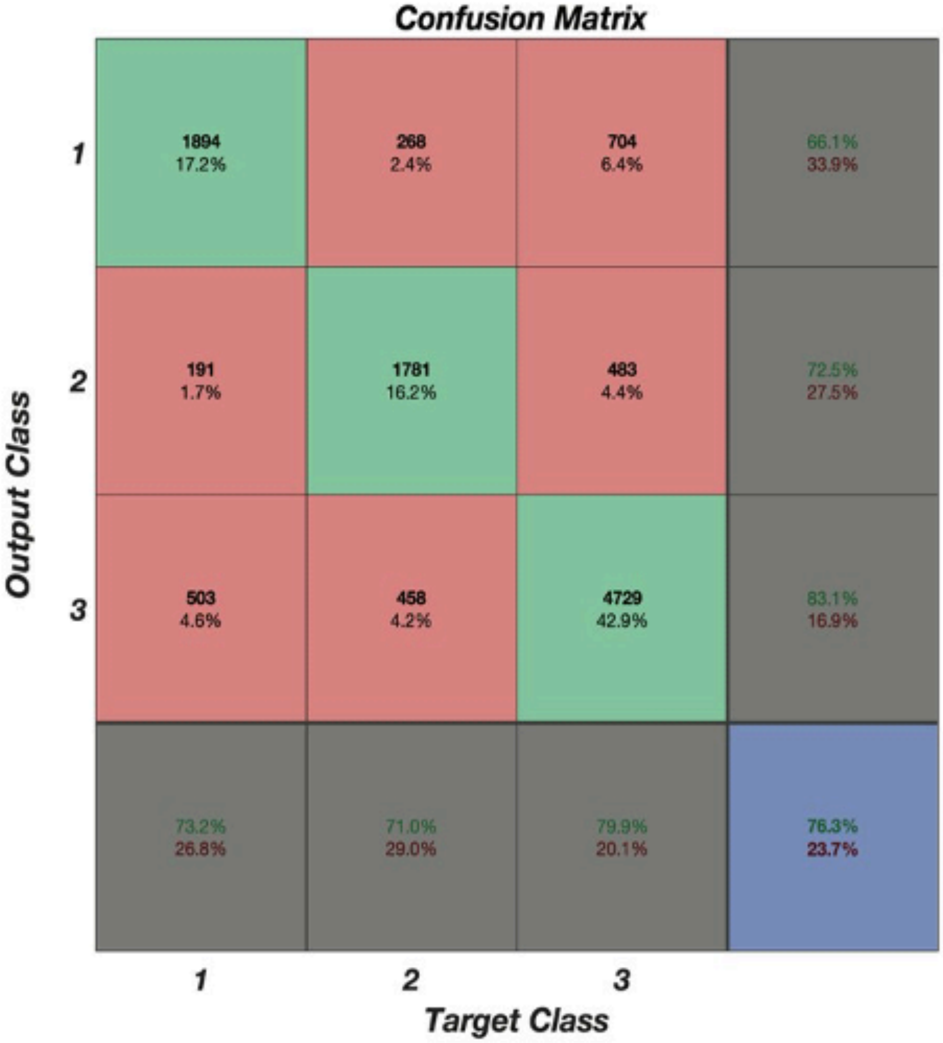


Figure 71 Topology assignments in the testing data set

6.2.1.1 Confusion matrix

The TMB topology prediction accuracy is 83.1%. Figure 72 represents the confusion matrices for the cascading classifier. The diagonal cells show the number of residue positions correctly classified for each topology class. The off-diagonal cells show the number of residue positions that were misclassified. The diagonal cells correspond to observations that are correctly classified. The number of observations and the percentage of the total number of observations is shown in each cell. The column on the far right of the plot shows the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. These metrics are often called the precision (or positive predictive value) and false discovery rate. The row at the bottom of the plot shows the percentages of all the examples belonging to each class that are correctly and incorrectly classified. These metrics are called the recall (or true positive rate) and false-negative rate. The cell in the bottom right of the plot shows the overall accuracy.



Confusion Matrix

Output Class	1	2	3	
1	1894 17.2%	268 2.4%	704 6.4%	66.1% 33.9%
2	191 1.7%	1781 16.2%	483 4.4%	72.5% 27.5%
3	503 4.6%	458 4.2%	4729 42.9%	83.1% 16.9%
	73.2% 26.8%	71.0% 29.0%	79.9% 20.1%	76.3% 23.7%
	1	2	3	

Target Class

Figure 72 Confusion Matrix

6.2.1.2 Receiver operating characteristic curve

The receiver operating characteristic for each output class is plotted with `Plotroc` (targets, outputs) and is represented in figure 73. When the curve goes to the left and top edges of the plot, the classification is better. The sensitivity measures the proportion of actual positives that are correctly identified as such. The false positive is also known as the fall-out. The fall-out is closely related to the specificity and is equal to (1 - specificity). Figure 73 represents the ROC curves for the cascading classifier.

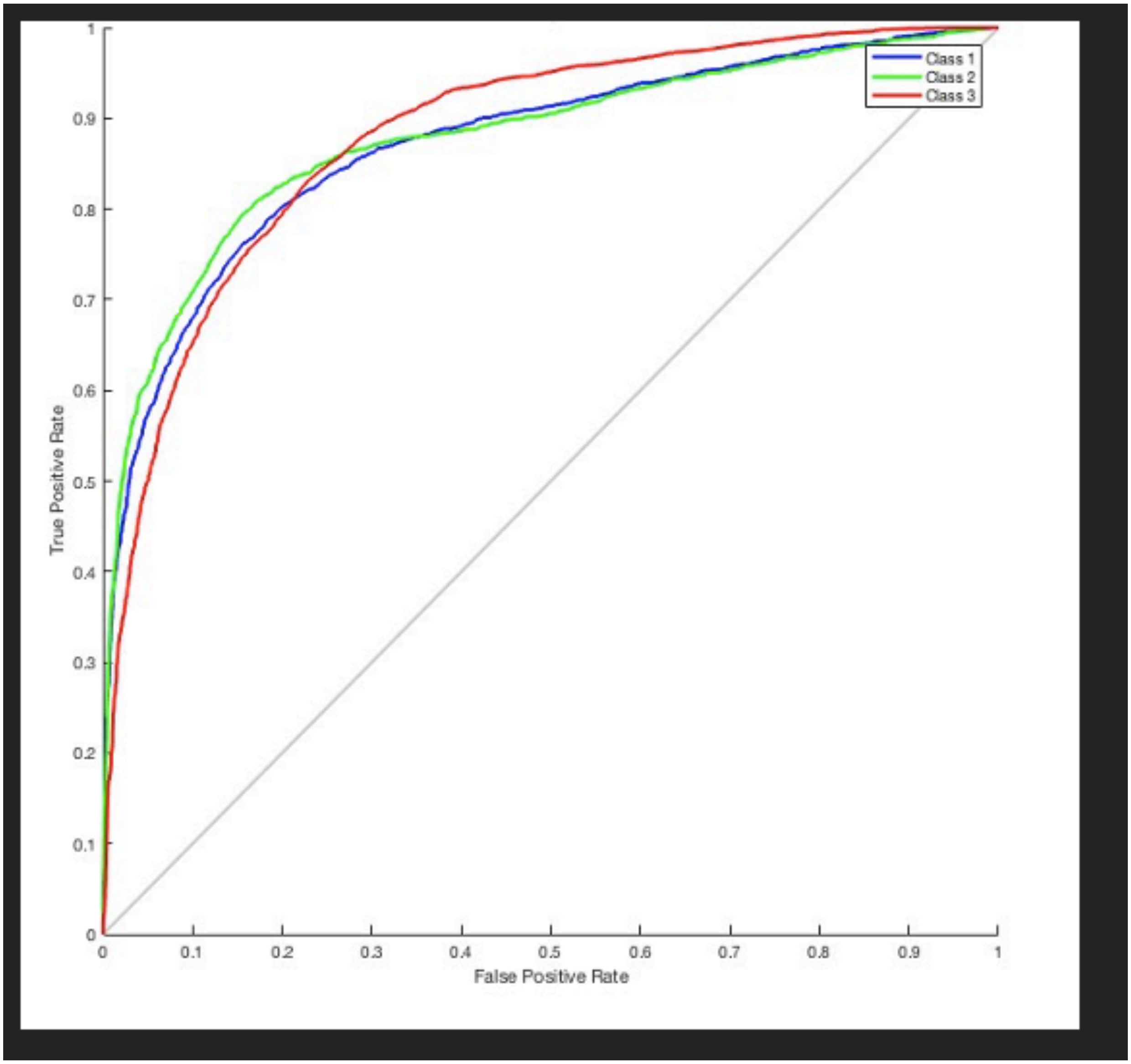


Figure 73 ROC curve

6.2.1.3 Prediction quality indices

The topology prediction is evaluated in detail by calculating prediction quality indices represented in figure 74. They indicate how well a particular state is predicted and whether overprediction or underprediction has occurred. The index `pcObs(S)` was defined for state S (S = {I, O, M}) as the number of residues correctly predicted in state S, divided by the number of residues observed in state S. Similarly, the index `pcPred(S)` for state S was defined as the number of residues correctly predicted in state S, divided by the number of residues predicted in state S. These quality indices are useful for the interpretation of the prediction accuracy.

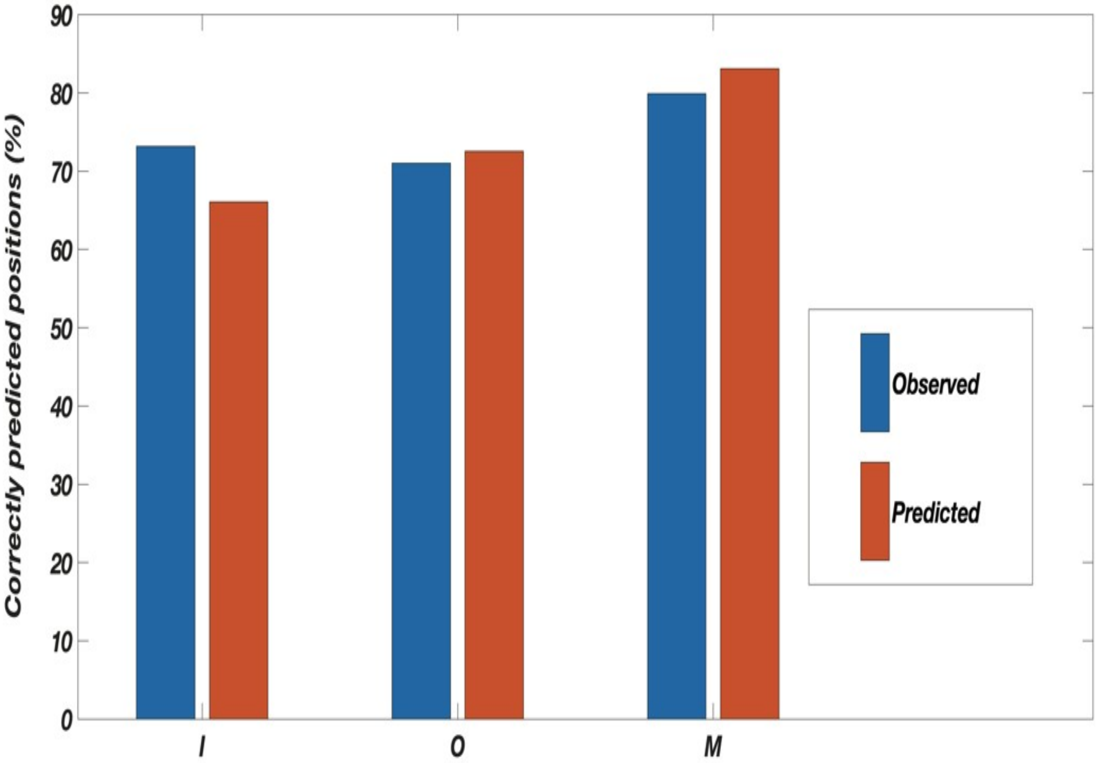


Figure 74 Quality indices performance

6.3 Summary

Chapter 6 focused on the cascading classifier implementation. Data division for the cascading classifier was defined by the model. Data division was based on two parameters corresponding to the fraction of the first level training set and the second-level training set.

The model consisted of two levels. Several selected models were trained at the first level. The selected algorithms (KNN, SVM or DNN) were trained to predict the values of the class. In case two or more classifiers were selected, a second level SVM classifier was trained to predict the value of the class based on the probability predicted by those two or more models at the first level. The model allowed to use a single method, two methods or even three methods. KNN =1 or 0, SVM = 1 or 0, and DNN = 1 or 0 are the list of parameters.. All combinations were possible except for combinations when they were equal to zero. If only one of them was set to one, the application worked as one classifier. If at least two parameters were set to one, several classifiers were trained and then combined by one more probability classifier. When multiple classifiers were chosen, a second level SVM was trained. The model had four different basic configurations. The parameters of each machine learning algorithm were modified. Multiple runs were executed in MATLAB using different parameters configurations. Best results are obtained with parameters configured to a window size of 55, hidden layer size of 55, log sigmoid transfer function, scaled conjugate gradient for the training function, Mean squared normalised error performance function and data division into three sets using random indices for the ANN part of the cascading classifier. *k*-value of eleven, exhaustive nearest neighbour search method, the random tie-breaking algorithm provided the best results for the KNN part of the cascading classifier. For the SVM part of the cascading classifier, a polynomial kernel function provided the best results. Overall, by constructing and using various machine-learning frameworks as part of the cascading classifier, a system has been developed and could predict the TMB topologies with significant robustness compared to other classifiers.

7. Discussion of findings

7.1 Implications

Chapter 5 discusses the artificial neural network implementation in MATLAB. The best overall accuracy when using the ANN-based approach is 63.4%. Putting efforts into finding an excellent neural network architecture will likely lead to better prediction accuracy. Training procedures are also of importance. One way to improve the performance, also applicable to other modern nonlinear machine learning techniques, is to improve data quality. The quality of models is generally constrained by the quality of the training data. You want the best data you can get for the problem. It is also good to have more data. The issue with TMB proteins is that there are limited datasets available, so a challenge of this project was trying to get the best accuracy while using small datasets. Deep learning usually gets better with more data. More data does not always help, but it can help. The performance of the artificial neural network was improved by checking for overfitting. It is essential to ensure that the neural network does not overfit. Overfitting refers to the state when the model memorises values from the training data instead of learning from those training data. When the model receives data that was never seen before, the model cannot perform well on them. Few techniques were used to avoid overfitting, including early stopping. It precipitates the deep neural network training, leading to a reduction in error in the test set. Checking for overfitting was one way to improve the performance of the deep neural network. Hyperparameters tuning was another way.

Hyperparameters are values that must be initialised to the network. Examples include the number of layers in the neural network, the activation function used, the training function or the performance function. Each neural network has a set of hyperparameters that can generate the best accuracy. There is no specific method for identifying the best collection of hyperparameters. It is mainly achieved with trial and error. Best practices can be used for some hyperparameters. For performance function, cross-entropy is preferred for classification, while mean squared error is one of the best choices for regression. The nonlinear functional inputs to the outputs are mapped with activation functions. They are essential, and choosing the proper activation function helps the model learn better. Log sigmoid function have been proven to provide the best accuracy for the implementation. The training function `trainscg`, which corresponds to the scaled conjugate gradient backpropagation, has been proven to provide the best accuracy for the implementation. It offers faster training with excellent test efficiency.

Chapter 5 also describes the implementation of the KNN in MATLAB. The best overall accuracy when using KNN based approach is 71.8%. The K-Nearest neighbors algorithm, KNN for short, is a classic machine learning algorithm that is often overlooked by deep learning. KNN is a supervised machine learning algorithm that is simple to implement yet can make robust classifications. One of the most significant advantages of KNN is that it is a lazy learner. It means that the model requires no training and get right to the classifying data. For the classification of some given data point, p, a KNN model compares first p to every other point available in its database using some distance metric. A distance metric is, for example, the Euclidean distance. It is a simple function that takes two points and returns the distance between them. Thus, it can be assumed that two points with a smaller distance between them are more similar than two points with a larger distance between them. This is the general idea behind KNN. k is some arbitrary value selected (usually between 3–11) that tells the model how many most similar points to p it should consider when classifying p. The model will then take those k most similar values and use a voting technique to classify p. With KNN, there is no explicit training phase, or it is very minimal. This also means that the training phase is fast. Lack of generalisation means that KNN keeps all the training data. All (or most) the training data is needed during the testing phase, to be more exact. Most KNN models use Euclidean or Manhattan distance as the go-to distance metric. These metrics are simple and perform well in a wide variety of situations. The Euclidean distance has been proven to provide the best accuracy for the implementation. It is the default distance metric in MATLAB when not specified. KNN performs much better if all the data have the same scale. KNN works well with a small number of input variables (p) but struggles when inputs are very large. In case of a tremendous value of k, it may be possible to include points from other classes in the neighborhood. In the case of the too-small value of k, the algorithm is very sensitive to noise. A value of k=8 has been proven to provide the best accuracy for the implementation. The optimal value for k was not chosen by cross-validation but rather by experimentation.

Chapter 5 also refers to the implementation of SVM in MATLAB. The best overall accuracy when using SVM based approach is 64.8%. Support Vector machines, so-called SVM, is supervised learning algorithm. It is used for classification and regression problems. It is usually used for smaller datasets. SVM is based on the idea of finding a hyperplane that best separates the features into different domains. The idea of SVM is to come up with an optimal hyperplane that will classify the different classes. The points closest to the hyperplane are called the support vectors points, and the distance of the vectors from the hyperplane are called the margins. The farther the support vector points are from the hyperplane; the higher is the probability of correctly classifying the points in their respective region or classes. There are a few advantages for SVMs. They are effective in the higher dimensions. It is the best algorithm when classes are separable. The hyperplane is affected by only the support vectors; thus, outliers have less impact. SVM is suited for extreme case binary classification. There are also a few disadvantages. A larger dataset requires a large amount of time to process. They do not perform well in case of overlapped classes, and selecting the appropriate kernel can be tricky. SVMs were initially designed for binary classification. The multi-class classification problem was decomposed into a series of binary problems such that the standard SVM can be directly applied. The implementation of SVM in MATLAB used `t= templateSVM ()` and `fitecoc` functions. `fitcecoc` function from the Statistics and Machine Learning Toolbox™ was used to create a multiclass classifier using binary SVMs. `fitcecoc` uses K (K – 1)/2 binary support vector machine (SVM) models using the one-versus-one coding design, where K is the number of unique class labels (levels). `fitcecoc` combines multiple binary learners using a coding design. By default, `fitcecoc` applies the one-versus-one design, which specifies training binary learners based on observations from all combinations of pairs of classes. For example, in a problem with ten classes, `fitcecoc` must train 45 binary SVM models.

Parameters of the SVM have been tuned to improve the performance. Parameter C represents the error penalty for misclassification for SVM. It maintains the trade-off between smoother hyperplane and misclassifications. Some misclassifications are allowed to avoid overfitting the classifier. The parameter C that trades how much misclassification of isolated points modifies the decision boundary is defined as 'BoxConstraint'. This parameter is specified as a numeric value and has been changed. A strategy for BoxConstraint is to try a geometric sequence of the box constraint parameter. For example, choose 11 values, from 1e-5 to 1e5, by a factor of 10. Increasing BoxConstraint might decrease the number of support vectors and might increase training time. Making the SVM a soft border algorithm considerably reduces the training time.

A unique strength of an SVM is the use of kernel function to map the data into a higher dimensional feature space. In training SVM, kernels and their parameters have a vital role in classification accuracy. Therefore, a suitable kernel design and parameters should be used for SVM training. Various kernels were used. The most efficient kernel was the polynomial kernel with polynomial order 3. It is aligned with the findings of the literature on this topic. The polynomial degree parameter controls the flexibility of the decision boundary. Higher degree kernels yield a more flexible decision boundary.

It was discussed earlier about overfitting in KNN. For SVM, to avoid overfitting, a soft Margin needs to be chosen instead of a hard one. For example, some data points can enter the margin intentionally so that the classifier doesn't overfit the training sample. Gamma (γ) is an important parameter, and it controls overfitting in SVM. Gamma is not technically an SVM hyperparameter. It is a parameter of the Kernel. Gamma (γ) is referred to as 'KernelScale' in MATLAB. The higher the gamma, the higher the hyperplane tries to match the training data. Therefore, choosing an optimal gamma to avoid overfitting and underfitting is the key. Increasing gamma leads to overfitting as the classifier tries to fit the training data perfectly. Various Kernelscale have been used, and the best results are obtained with KernelScale=10. Error penalty (Parameter C/ BoxConstraint), kernel (Kernel) and regularisation (Gamma/ kernelScale) are the most important hyperparameters that have been modified for the SVM.

Chapter 6 finally discusses the implementation of a cascading classifier in MATLAB. There's been increasing use of ensemble learning methods in recent research in computational biology. Ensemble learning combines multiple learning algorithms to improve the overall prediction accuracy. It is one of the most promising solutions for many biological problems. Cascading is a specific case of ensemble learning. It is based on the concatenation of several classifiers using the information provided from the output of a given classifier as additional information for the next classifier in the cascade. The model consisted of two levels. Several selected models were trained at the first level. The selected algorithms (KNN, SVM or ANN) were trained to predict the values of the class. In case two or more classifiers were selected, a second level SVM classifier was trained to predict the value of the class based on the probability predicted by those two or more models at the first level. This is a cascading classifier as the output of the first layer corresponds to the input of the second layer. If only one model is initially selected at the first level, the second layer classifier was not trained. The model allowed to use a single method, two methods or even three methods. KNN =1 or 0, SVM = 1 or 0, and ANN = 1 or 0 were the list of parameters to select to create a specific cascading classifier. If a parameter were equal to 0, it would not be used in the cascading classifier. Optimisation of the cascading classifier was mostly obtained by optimising each machine learning algorithm used and by starting using the parameters that gave the best results for each machine learning algorithm. The best overall accuracy when using the cascading classifier approach was 76.3%, including a TMB topology prediction of 83.1%. The accuracy of 83.1% is for one scenario combination where layer one includes SVM, KNN and ANN, and layer two include SVM. Many important biological processes, such as cell signalling, transport of membrane-impermeable molecules, cell-cell communication, and cell recognition and adhesion, are mediated by membrane proteins. The methodology evaluated and created as part of this dissertation could be applied to any TMB proteins and could potentially help identify new targets for antibiotics, vaccines, or antimicrobials. The application of the cascading classifier to TMB topology prediction has been published in two recent papers (Kazemian et al., 2018) and (Kazemian et al., 2020).

7.2 Limitations

One limitation of this research is that only small datasets have been used. In recent years, machine learning techniques have changed the world. It is easier nowadays to perform a variety of complex tasks much easier. Deep learning models can be more successful with a large amount of training data. ResNet (He et al., 2016) represents an architecture for image classification. It won the best place at the classification competition ILSVRC in 2015. ResNet has a deep and complex architecture, and it has been trained with about 1.2 million images. In the industry and academia, it is well agreed that different algorithms can perform almost the same when there is enough data for a given problem. It is essential to understand that the extensive data need to have meaningful information for the model to learn from that. Although machine learning techniques require less data than deep learning, extensive data similarly impacts performance. The cascading classifier presented as part of this research could have been evaluated using larger datasets. It is especially critical when the cascading classifier uses artificial neural networks as part of the architecture.

The datasets used in this research are small since there are only a few TMB proteins. However, SVM performs better with larger datasets and using larger datasets could have potentially increased accuracy. Sordo and Zeng (Sordo and Zeng 2005) investigated the dependency between the sample sizes and classification accuracies of three different classification techniques: Naïve Bayes, Decision Trees and Support Vector Machines. They used a set of 8500 text excerpts extracted automatically from narrative reports from the Brigham & Women's Hospital, Boston, USA. Their results confirm a correlation between the size of the training set and the classification. The algorithms perform well with small datasets. When the number of cases increases, Support Vector Machines and decision trees increases performance.

Another limitation to this research is that only binary encoding techniques have been used. Amino acid encoding plays a fundamental role in the final success of machine-learning-based protein structure and function prediction methods. Amino acid encoding (Different from the protein sequence encoding) can be utilised in both residue-level and sequence-level prediction of protein properties using combinations of different algorithms. The methods can be grouped into five categories based on the information sources and information extraction methods. It includes binary encoding, physicochemical properties encoding, evolution-based encoding, structure-based encoding, and machine-learning encoding.

Binary encoding is a directly encoding way that transforms each amino acid into a 20dimensional elementary unit vector (Chauhan, Rao, and Raghava, 2013). The Alanine [A], for example, maybe be encoded into (10000000000000000000) and Cysteine into (01000000000000000000). A peptide sequence of amino acid residues corresponds equally to an n*20-dimensional sparse representation. The binary encoding reflects the information on types and positions of residues in the protein sequence (Gnad, Ren, Choudhary, Cox, and Mann, 2010) (Huang and Li, 2017). A binary representation can be equally recovered into the corresponding amino acid sequence. Conversion of amino acid sequences into real numbers to get numerical input vectors is critical for constructing models. In practice, 20 binary bits are the most common distributed encoding method, in which each amino acid is represented by a unique 20-bit binary string consisting of nineteen 0s and one 1, since there are 20 amino acids (Qian and Sejnowski,1988), (Yang and Chou,2004). Bose et al. (Bose, Subrata K., et al. 2007) result also suggested that 20-bit binary encodings can achieve a higher classification accuracy and robustness compared with 5-bit and 9-bit encodings, indicating that the use of simple physicochemical parameters may not increase the robustness of the system more so than the binary encodings. In the sliding window method, a window becomes one training pattern for predicting the topology of the residue at the centre of the window. Information about local interactions among neighboring residues is embedded in each training pattern. The feature value of each amino acid residue in a window represents each residue's weights (costs) in a pattern.

Physicochemical properties encoding techniques have been discussed in this research, but it was not considered. Physicochemical property such as the hydrophobicity of amino acid seems to have an essential role in the organisation of the self-assembly of proteins. Apart from hydrophobicity properties, the codon diversity and the size of amino acids can be used as features. The codon diversity of amino acids corresponds to the number of codons coding for the amino acid. The size of the amino acids reflects their molecular volume. Regarding physicochemical properties encodings, the variety of properties and the extraction methodologies are essential factors in building a valuable encoding.

Structure-based amino acid encoding methods, which can be named statistical-based methods, can encode amino acids using the structure-related statistical potentials using the inter-residue contact energies (Tanaka and Scheraga, 1976). They have not been considered in this research. Structure-based encoding methods had application in protein secondary structure prediction and protein fold recognition. The structural potential of amino acids, linked to the protein structure and function, is reflected by those encodings. More and more proteins have known form, and the use of structure-based encodings is becoming very useful. For protein function prediction, encodings reflecting function potentials are helpful. This critical topic looks at function-based encoding methods that could be further explored.

8. Conclusion and future research directions

This research answered the following questions: What are the best machine learning techniques for the topology prediction of TMB proteins? How can the prediction for the topology prediction of TMB proteins be improved?
The research approach has been classified as an experimental study in evaluating multiple machine learning approaches, including artificial neural network, KNN, SVM and cascading classifier for the topology prediction of TMB proteins. A procedure has been intentionally introduced. Results and outcomes have been observed and discussed. The approach involved a controlled and systematic procedure in minimising error and bias. Another critical element related to this experiment is the use of random assignment. An artificial neural network, a KNN, SVMs and a cascading classifier have been created and implemented in MATLAB. The boctopus2 dataset (Hayat, Peters, Shu, Tsirigos and Elofsson, 2016) and TOPDB dataset (Tusnady, Kalmár and Simon, 2008) have been used. The performances of the various machine learning techniques have been evaluated. This research dealt with the prediction of TMB topologies, one of the most significant problems in structural molecular biology. Using a cascading classifier such as the one developed as part of the thesis is recommended for TMB topology prediction.

It is also essential to discuss the merits and shortcomings of this research thesis. When using the experimental research defined in this thesis, specific hyperparameters have been isolated, and therefore it was possible to determine if a potential outcome is viable. Each hyperparameter was controlled independently or in different combinations to study what possible outcomes were available for the theory. This provided an advantage in the ability to find accurate results. The research offered specific conclusions. Because experimental research provides such a high level of control, it produced specific and relevant results with consistency. It was possible to determine success or failure, making it possible to understand the validity of the models developed in a much shorter amount of time than other verification methods. The research thesis allows for its duplication if others control the same hyperparameters. This helps to promote the validity of the models. The manipulation of variables enables any researcher to look at various cause-and-effect relationships produced by the models. It allows researchers to look deeper into the possibilities, indicating how the different variable relationships can provide unique benefits. Experimental research requires unique levels of variable control, and there is a high risk of experiencing a human error during the study. An error could eliminate the validity of the experiment. The research needed to isolate various variables and conduct testing on it. This process was lengthy and required a lot of research and preparation, primarily for data preparation. This would not be very convenient if the methodology were used in an industrial environment.

The use of larger datasets is one of the possible directions for future research. A recent article (Ajiboye et al., 2015) evaluated the effect of the sizes of datasets on the predictive model with the use of supervised machine learning. The authors examined the predictive model's capability to generalise a particular dataset size when simulated with new untrained input. The article discusses the experiment using three different sizes of data and the MATLAB program to create predictive models to determine if the size of data influences the model accuracy. The measurement of the simulated output of each model has been executed using the Mean Absolute Error (MAE). Comparisons have been made. The article indicates that the quantity of data used for training must represent the entire sets and be sufficient to span through the input space. The simulation of the three network models indicated that the learning model using the largest training sets appears to be the most accurate and delivers better and more stable results consistently.

Before starting to understand how more data improves the performance of a model, it is essential to understand bias and variance—considering a dataset including a quadratic relationship between independent and dependent variables and that the genuine relationship is not known and approximated linearly. In that situation, a significant difference between the prediction of the model and the actual observed data will be seen. The difference that happens between the observed value and the predicted value is named bias. Those models are considered to have less power and represent underfitting. In the example chosen, when approximating the relationship as cubic or higher powers, there is a case of high variance. Variance can be defined as the difference in performance on the training set compared to the test set. One issue regarding high variance is that models fit the training data well, but they do not generalise well on datasets out of training. Validation and test set are therefore critical when building the models. The goal is usually to minimise variance and bias, for example, making a model that fits the training data well and a model that can generalise well on test data and validation data. Lots of techniques are available to achieve this. Training with many data is one of the ways to accomplish this goal. Brain and Webb (Brain and Webb, 2000) looked at the impact of the size of datasets on variance and bias for classification problems. Their observation is that variance will decrease as the training dataset size increases. No comments were provided on the bias.

Statistical methods used to estimate the dataset size requirements and the classification of microarray data using learning curves are suggested in an article written by Mukherjee et al. (Mukherjee et al., 2003). The article pays particular attention to using the existing classification results to estimate dataset size requirements for future classification experiments. The study evaluated the increase in accuracy and significance of classifiers built with additional data. The paper concluded that the subsampling procedure gives more accurate estimates of the quantiles of the true error of a classifier as the number of subsamples increases.

Future work for this paper could be extended using various encoding techniques. A recent article by Jing et al. (Jing, Dong, Hong, and Lu, 2019) provides a detailed assessment of the different amino acids encoding techniques available. Their study indicates that the evolution-based position-dependent encoding method PSSM offers the best performance. Structure-based and machine-learning encoding methods provide the potential for further application, especially the neural network-based distributed representation of amino acids. Evolution-based encoding could be considered for future research. Evolution-based encoding methods extract evolutionary information of residues from sequence alignments or phylogenetic trees to represent amino acids. It uses mainly the amino acids substitution probability. Those methods can be divided into two main groups based on the position relevance: There are position-independent methods, such as the Point Accepted Mutation (PAM) matrix and the BLOSUM matrix. There are also position-dependent methods. The position-independent methods encode amino acids using the fixed encodings independently of the amino acid position in the sequence and the amino acid composition of the series. For the position-dependent methods, amino acids are encoded at different positions using different encodings, even if the amino acid types are the same. PSMM (Position-Specific Scoring Matrix) is a prevalent encoding method. Position-Specific Scoring Matrix (PSSM) used the position-specific scoring matrix that has been provided by PSI-BLAST (Kim and Park, 2003). Individual profiles reflect detailed conservation of amino acids in a family of homologous proteins in this coding. This approach was used first by Jones (Jones,1999) to predict protein secondary structure using a neural network. Based on the author results, PSI-BLAST is a highly efficient sequence query method. This is due to three different aspects. First, the alignments used by PSI-BLAST are based on pairwise local alignments. A previous study written by Salamov and Solovyev (Salamov and Solovyev, 1997) indicated that the prediction accuracy could be increased when using reliable local assignments. Secondly, based on the iterated profiles, the sensitivity of PSI-BLAST has been increased. Thirdly, authors have been using various automatic multiple sequence alignments. Among all the alignments used, the PSI-BLAST alignments were the best performer.

Machine learning encoding could be considered for future research. It is very different from manual encoding techniques and cannot be compared. Amino acid encodings are learned from the protein sequence or structure data by machine learning based encodings techniques with the use of machine learning techniques such as artificial neural networks. To reduce the complexity of the model, the neural networks share weight for twenty amino acids. The input layer is the original encoding of the target amino acid. It can be one-hot encoding or physicochemical encoding, for example. The output layer is the original encoding of the related amino acids. The new encoding of the target amino acid, represented by the hidden layer, has a reduced dimension compared to the original encoding. The concept of learning-based amino acids encodings was introduced by Riis and Krogh (Riis and Krogh, 1996). They used a 20x3 weight sharing neural network to learn a 3-dimensional real numbers representation of 20 amino acids from the one-hot encoding to reduce the redundancy of one-hot encoding. For human signal peptide cleavage sites recognition, Jagla and Schuchhardt (Jagla and Schuchhardt, 2000) applied later the weight sharing artificial neural network to learn a 2-dimensional encoding of amino acids. Various new machine-learning-based encoding methods have been suggested referencing distributed word representation in natural language processing. The machine-learning encoding methods have great potential in future studies. Amino acid encoding is an open problem, and encoding methods are based on an artificially defined basis. For example, researchers have observed physicochemical properties encodings are constructed from protein fold-related properties. It will inevitably bring some unknown deviations. Those artificial deviations can be avoided by automatically learning the amino acid from biological data. Natural languages and protein sequences have similarities. Protein sequences can be looked at as sentences. The amino acid or polypeptide chain can be seen as words in languages. The term distributed representation has obtained more outstanding performances in natural language processing tasks. Protein sequences could also improve when using the distributed representations of amino acid or n-gram amino acids. Recent studies have indicated the potential of amino acid distributed representations in protein family classification, protein functional properties prediction and disordered protein identification. Most of these methods look at the ngram amino acid distributed representations and cannot be directly used for the residue-level properties prediction. More research is needed on the residue-level distributed representations of amino acids.

References

Ajanki, A. (2007). Example of k-nearest neighbors classification. [Online]. Available at: http://commons.wikimedia.org/wiki/File:KnnClassification.svg. [Accessed 13 Oct. 2016].

Ajiboye, A. R., Abdullah-Arshah, R., and Hongwu, Q. *Evaluating the effect of dataset size on predictive model using supervised learning technique*. 2015

Ahn, C. S., Yoo, S. J., Park, H. S. (2003). Prediction for beta-barrel Transmembrane Protein region using HMM. *Journal of the Korea Information Science Society*, 30, 802–804.

Allwein, E. L., Schapire, R.E and Singer, Y. (2000). Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, 1,113–141.

Amaratunga D., Cabrera J., Lee Y. (2008). Enriched random forests. *Bioinformatics.* 24, 2010– 2014.

Amari, S. and Wu, S. (1999). Improving support vector machine classifiers by modifying kernel functions. *Proceedings of International Conference on Neural Networks*, 12, 783-789.

Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, I. H., Zhang, Z., Miller, W., and Lipman, D. I. (1997). Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25, 3389 3402.

Armananzas R., Inza I., Larranaga P. (2008). Detecting reliable gene interactions by a hierarchy of Bayesian network classifiers. Computer Methods and Programs in Biomedicine. 91,110–121.

Bagos, P. G, Liakopoulos, T. D, Spyropoulos I. C, Hamodrakas S. J. (2004a). A Hidden Markov Model method can predict and discriminate beta-barrel outer membrane proteins. *BMC Bioinformatics*, 5, 29

Bagos, P. G, Liakopoulos, T. D, Spyropoulos I. C, Hamodrakas S. J. (2004b). PRED-TMBB: a web server for predicting the topology of beta-barrel outer membrane proteins. *Nucleic Acids Research*, 32, W400-W404.

Bagos, P. G., Liakopoulos, T. D. and Hamodrakas, S. J. (2005). Evaluation of Methods for Predicting the Topology of Beta-Barrel Outer Membrane Proteins and a Consensus Prediction Method. *BMC Bioinformatics*, 6, 7.

Baldi, P., Pollastri, G. (2002) A machine-learning strategy for protein analysis. IEEE Intell. Syst. 17(2), 28–35

Bassiou, N., Kotropoulos, C., Kosmidis, T. and Pitas, I. (2001). Frontal Face Detection Using Support Vector Machines and Back-Propagation Neural Networks. *Proceedings 2001 International Conference on Image Processing,*1,1026–1029 BCA Chemistry [Online]. Available at: https://bcachemistry.wordpress.com/tag/spider-silk/
[Accessed 07 Mar. 2016]

Beale, E.M.L. (1972). A derivation of conjugate gradients. *Numerical methods for nonlinear optimisation*. (Ed. D. A. H. Jacobs), Academic Press, London.

Bengio, S., and Mariethoz, J. (2001). Learning the Decision Function for Speaker Verification. *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings,* 1, 425–428.

Bhanot G, Alexe G, Venkataraghavan B, Levine AJ. (2006). A robust meta-classification strategy for cancer detection from MS data. *Proteomics.*6, 592–604.

Bigelow, H. R., Petrey D. S., Liu J., Przybylski, D., Rost, B. (2004). Predicting transmembrane beta-barrels in proteomes. *Nucleic Acids Research,* 32, 2566–2577

Bigelow, H., and Rost, B. (2006). PROFtmb: A Web Server for Predicting Bacterial Transmembrane Beta Barrel Proteins. *Nucleic Acids Research,* 34, W186–188.

Bin, Z., Yong, L. and Shao-Wei, X. (2000). Support Vector Machine and Its Application in Handwritten Numeral Recognition. *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, 2,720–723

Bonab, H.R. and Can, F. (2016). A Theoretical Framework on the Ideal Number of Classifiers for Online Ensembles in Data Streams." In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, 2053–56. CIKM '16. New York, NY, USA: ACM

Bose, Subrata K., et al. "Use of Artificial Neural Networks and Effects of Amino Acid Encodings in the Membrane Protein Prediction Problem." Progress in Pattern Recognition (2007): 37.

Boser, B. E., Guyon, I. and Vapnik, V. (1992). A Training Algorithm for Optimal Margin Classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory,*144152, ACM Press, Pittsburgh

Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Jackel, L.D, LeCun, Y. (1994). Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing*, 2, 77–82

Brain, D. and Webb, G. *On the Effect of Data Set Size on Bias and Variance in Classification Learning* Proceedings of the Fourth Australian Knowledge Acquisition Workshop, Jun. 2000

Breiman, L. (1994). Bagging predictors. *Machine Learning*, 24, 123–140

Brown, G., Wyatt, J., Harris, R. and Yao, X. (2005). Diversity creation methods: a survey and categorisation., Information Fusion, 6, 5-20.

Bryman, A. & Bell, E. (2007). Business Research Methods, 2nd edition. Oxford University Press.

Brusic V., Rudy G., and Harrison L.C. (1995). Prediction of MHC binding peptides using artificial neural networks. Complexity International volume 2, ISSN 1320-0682.

Burges, C.C. (1998). A tutorial on support vector machines for pattern recognition. *Proceedings of International Conference on Data Mining and Knowledge Discovery*, 2,121-167.

Burges, C.J.C. (1998) A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2, 121-167.

Byun, H., Lee, S. W. (2003). Applications of support vector machines for pattern recognition: a survey. *Pattern recognition with support vector machines*. *Lecture Notes in Computer Science*, 2388, 571–591

Campbell, I.C.G. (2000). An introduction to kernel methods In R. J. Howlett, & L. C. Jain (Eds.), Radial Basis Function Networks: Design and Applications,155-192, Springer-Verlag, Berlin.

Caragea, C., Sinapov, J., Silvescu, A., Dobbs, D. and Honavar, V. (2007). Glycosylation Site Prediction Using Ensembles of Support Vector Machine Classifiers. *BMC Bioinformatics* 8, 438.

Chauhan, J. S., Rao, A., and Raghava, G. P. S. “In silico Platform for Prediction of N-, O- and CGlycosites in Eukaryotic Protein Sequences,” *PLOS ONE*, vol. 8, no. 6, p. e67008, Jun. 2013.

Cheng, C.-T., Feng, Z.-K., Niu, W.-J. and Liao, S.-L. (2015). Heuristic Methods for Reservoir Monthly Inflow Forecasting: A Case Study of Xinfengjiang Reservoir in Pearl River, China. *Water*, [online] 7(12), pp.4477–4495.

Chetwynd, A. P., Scott, K. A., Mokrab, Y. & Sansom, M. S. P. (2008). CGDB: a database of membrane protein/lipid interactions by coarse-grained molecular dynamics simulations. *Molecular Membrane Biology*, 25, 662–9.

Chou, P. Y. and Fasman, G. D. (1974a). Conformational parameters for amino acids in helical, b-sheet, and random coil regions calculated from proteins. *Biochemistry*, 13, 211-221.

Chou, P. Y. and Fasman, G. D. (1974b). Prediction of protein conformation. *Biochemistry*, 13, 222–245

Cortes, C., Vapnik, V. (1995). Support vector networks. Mach. Learn. 20(3), 273–297

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. Information Theory, *IEEE Transactions on information theory*, 1,21–27.

Cowan, S. W., Schirmer, T. (1992). Crystal Structures Explain Functional Properties of Two E. Coli Porins. *Nature,* 358, 727-733.

Cserzo, M., Wallin, E., Simon, I., von Heijne, G. and Elofsson, A. (1997). Prediction of transmembrane alpha-helices in prokaryotic membrane proteins: the Dense Alignment Surface method. *Protein Engineering, 10*, 673-676

Daniel, A. (2003). Prediction of Protein Secondary Structure using Long Short-Term Memory. Technical report, Halmstad University, 1-35.

Deng, L., Guan, J., Dong, Q. and Zhou, S. (2009). Prediction of Protein-Protein Interaction Sites Using an Ensemble Method." *BMC Bioinformatics,* 10, 426.

Dettling M. (2004). BagBoosting for tumor classification with gene expression data. *Bioinformatics.* 20,3583–3593

Diederichs, K., Freigang, J., Umhau, S., Zeth, K. and Breed, J. (1998). Prediction by a neural network of outer membrane beta-strand protein topology. *Protein Science,* 7, 2413–2420.

Dietterich, T.G., and Bakiri, G. (1995). Solving Multiclass Learning Problems via ErrorCorrecting Output Codes. *Journal of Artificial Intelligence Research,* 2, 263–286.

Dos Santos, E.M. and Gomes, H. M. (2002). A Comparative Study of Polynomial Kernel SVM Applied to Appearance-Based Object Recognition. *Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines*, p.408-418.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, 14,179–211

Elmhurst College-Secondary Protein-Structure [Online]. Available at: http://elmhurst.edu/~chm/vchembook/566secprotein.html [Accessed 07 Mar 2016].

Enzim.hu. (2014). TOPDB. [online] Available at: http:/topdb.enzim.hu [Accessed 12 Nov. 2021].

Escalera, S., Tax, D.M.J., Pujol, O., Radeva, P. and Duin, R.P.W. (2008). Subclass Problem-Dependent Design for Error-Correcting Output Codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.30(6), pp.1041–1054

Escalera, S., Pujol, O. and Radeva, P. (2009). Separability of Ternary Codes for Sparse Designs of Error-Correcting Output Codes. *Pattern Recognition Letters* 30, 285–297.

Escalera, S., Radeva, P., & Pujol, O. (2008). Coding and decoding design of ECOCs for multiclass pattern and object recognition. Universitat Autònoma de Barcelona.

Fan, R. E., Chen, P.H. and. Lin, C. J. (2005). Working set selection using second order information for training support vector machines. Journal of Machine Learning Research, 6,1889–1918

Fariselli, P., Martelli, P. L. and Casadio, R. (2005). A New Decoding Algorithm for Hidden Markov Models Improves the Prediction of the Topology of All-Beta Membrane Proteins. *BMC Bioinformatics* 6, 4.

Fletcher, R., and Reeves, C.M. (1964). Function minimization by conjugate gradients. *Computer Journal,* 7, 149-160.

Freeman, T. C. and Wimley, W. C. (2012). TMBB-DB: A Transmembrane B-Barrel Proteome Database. *Bioinformatics (Oxford, England)* 28, 2425–2430.

Ganapathiraju, A., Hamaker, J., Picone, J. (2004). Applications of support vector machines to speech recognition. IEEE Trans. Signal Process. 52(8), 2348–2355

Garrow, A. G., Agnew, A. and Westhead, D. R. (2005). TMB-Hunt: A Web Server to Screen Sequence Sets for Transmembrane Beta-Barrel Proteins. *Nucleic Acids Research*, 33, W188–92

Gnad, F., Ren, S., Choudhary, C., Cox, J. and Mann, M. "Predicting post-translational lysine acetylation using support vector machines," *Bioinformatics*, vol. 26, no. 13, pp. 1666–1668, Jul. 2010

Goldberg, Y., Elhadad, M. (2008). SplitSVM: fast, space-efficient, non-heuristic, polynomial kernel computation for NLP applications. *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*.

Golik, P., Doetsch, P., Ney, H., 2013. *Cross-entropy vs squared error training: a theoretical and experimental comparison*, In Interspeech-*2013*, 1756-1760.

Gordon, J. J., Towsey, M. W., Hogan, J. M., Mathews, S. A. and Timms, P. (2006). Improved Prediction of Bacterial Transcription Start Sites. *Bioinformatics,* 22, 142–148.

Gorgevik, D., Cakmakov, D. and Radevski, V. (2001). Handwritten Digit Recognition by Combining Support Vector Machines Using Rule-Based Reasoning. *Proceedings of the 23rd International Conference on Information Technology Interfaces,1*, 139–144

Graves, A. (2011). Practical variational inference for neural networks. Neural Information Processing Systems Conference, 2011

Graves, A. (2012). Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence. Springer

Gromiha, M. M. and Ponnuswamy, P. K. (1993). Prediction of transmembrane beta-strands from hydrophobic characteristics of proteins. *International Journal of Peptide and Protein Research*, 42, 420–31.

Gromiha, M. M., Majumdar, R., Ponnuswamy, P. K. (1997). Identification of membrane-spanning beta strands in bacterial porins. *Protein Engineering,*10, 497–500.

Gromiha, M. M., Ahmad, S., Suwa, M. (2004). Neural network-based prediction of transmembrane beta-strand segments in outer membrane proteins. *Journal of computational chemistry*, 25, 762–767.

Gromiha, M. M., Ahmad, S., and Suwa, M. (2005). TMBETA-NET: Discrimination and Prediction of Membrane Spanning Beta-Strands in Outer Membrane Proteins. *Nucleic Acids Research*, 33, W164–167.

Gromiha, M. M. (2010). *Protein Bioinformatics: From Sequence to Function*. Academic Press.

Guan Y., Myers C.L, Hess D.C, Barutcuoglu Z., Caudy A.A, Troyanskaya O. (2008). Predicting Gene Function in a Hierarchical Context with an Ensemble of Classifiers. *Genome Biology*, 9 Suppl 1

Guo, G., Li, S. Z. and Chan, K. (2000). Face Recognition by Support Vector Machines. *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, 196–201.

Guo, G., Li, S. Z. and Chan, K. (2001). Support vector machines for face recognition. *Journal of Image and Vision Computing*,19, 631-638

Guo, X., Alipour-Fanid, A., Wu, L., Purohit, H., Chen, X., Zeng, K. and Zhao, L. (2019). Multi-stage Deep Classifier Cascades for Open World Recognition. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*

Gutschoven, B., and Verlinde, P. (2000). Multi-Modal Identity Verification Using Support Vector Machines (SVM). *Proceedings of the Third International Conference on Information Fusion*, 2

Hassan M.R., Hossain M.M., Bailey J., Macintyre G., Ho J., Ramamohanarao K. (2009). A voting approach to identify a small number of highly predictive genes using multiple classifiers. BMC Bioinformatics.10, S19.

Hayat, S. and Elofsson, A. (2012). BOCTOPUS: improved topology prediction of transmembrane β-barrel proteins. *Bioinformatics*, 28, 516–522.

Hayat, S., Peters, C., Shu, N., Tsirigos, K. D. and Elofsson, A. (2016). Inclusion of Dyad-Repeat Pattern Improves Topology Prediction of Transmembrane Β-Barrel Proteins. *Bioinformatics (Oxford, England),* 32.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Haykin, S. (2009). *Neural Networks and Learning Machines Third Edition*. Upper Saddle River, New Jersey: Pearson Education Inc., 232–236.

He, K. Zhang, X., Ren, S. and Sun, J. *Deep Residual Learning for Image Recognition*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778

Heffernan, R., Paliwal, K., Lyons, J., Dehzangi, A., Sharma, A., Wang, J., Sattar, A., Yang, Y., and Zhou, Y. (2015). Improving Prediction of Secondary Structure, Local Backbone Angles, and Solvent Accessible Surface Area of Proteins by Iterative Deep Learning. *Scientific Reports,* 5, 11476.

Heisele, B., Ho, P., Wu, J. and Poggio, T. (2003). Face Recognition: Component-Based Versus Global Approaches. *Computer Vision and Image Understanding.* 91, 6–21.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.R., Jaitly, N., Senior, A. et al. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* 29, 82–97.

Hirokawa, T., Boon-Chieng, S. and Mitaku, S. (1998). SOSUI: classification and secondary structure prediction system for membrane proteins. *Bioinformatics*, 14, 378–379.

Holley, L.H. and Karplus, M. (1989). Protein secondary structure prediction with a neural network. *Proceedings of the National Academy of Sciences*, 86, 152-156.

Hsu, C.W., and Lin, C.J. (2002). A Comparison of Methods for Multiclass Support Vector Machines. *IEEE Transactions on Neural Networks* 13, 415–425.

Hu J., Yang Y.D., Kihara D. (2006). EMD: an ensemble algorithm for discovering regulatory motifs in DNA sequences. *BMC Bioinformatics.* 7, 342.

Huang, G., and Li, J. "Feature Extractions for Computationally Predicting Protein Post- Translational Modifications," *Current Bioinformatics*, vol. 12, Jul. 2017.

Hussey, J. and Hussey, R. (1997) Business Research: A Practical Guide for Undergraduate and Postgraduate Students. Macmillan, London.

Jacoboni, I., Martelli, P. L., Fariselli, P., De Pinto, V. and Casadio, R. (2001). Prediction of the transmembrane regions of beta-barrel membrane proteins with a neural network-based predictor. *Protein Science*, 10, 779–787.

Jaeger, H. (2001). The "Echo State" Approach to Analysing and Training Recurrent Neural Networks. Technical Report GMD Report 148, German National Research Center for Information Technology.

Jagla, B. and Schuchhardt, J. "Adaptive encoding neural networks for the recognition of human signal peptide cleavage sites," *Bioinformatics*, vol. 16, no. 3, pp. 245–250, Mar. 2000.

Jähnig, F. (1990). Structure Predictions of Membrane Proteins Are Not That Bad." Trends in *Biochemical Sciences* 15, 93–95.

Jaitly, N. and Hinton, G. E. (2013). Vocal tract length perturbation (VTLP) improves speech recognition. 30th International Conference on Machine Learning (ICML 2013)

Jayasinghe, S., Hristova, K. & White, S.H. (2001). MPtopo: A database of membrane protein topology. *Protein Science*, 10, 455–458

Jim, K.-C., Giles, C. L., and Horne, B. G. (1996). An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on Neural Networks*, 7, 1424–1438

Jing, X., Dong, Q., Hong, D. C., and R. Lu, "Amino acid encoding methods for protein sequences: a comprehensive review and assessment," *IEEE/ACM Trans Comput Biol Bioinform*, Apr. 2019.

Jones, D. T., Taylor, W. R. & Thornton, J. M. (1994). A model recognition approach to the prediction of all-helical membrane protein structure and topology. *Biochemistry*, 33, 3038–49.

Jones, D. T. "Protein secondary structure prediction based on position-specific scoring matrices," *J. Mol. Biol.*, vol. 292, no. 2, pp. 195–202, Sep. 1999.

Jonsson, K., Kittler, J., Matas, Y. P. (2002). Support Vector Machines for face authentication. *Image and Vision Computing*, 20, 369–375

Jordan, M.I. (1990). Attractor dynamics and parallelism in a connectionist sequential machine pages 112–127. IEEE Press.

Juwei Lu, K., Plataniotis, N., Venetsanopoulos, A. N. (2001). Face recognition using feature optimization and ν-support vector learning. *Neural Networks for Signal Processing XI: Proceedings of the 2001 IEEE Signal Processing Society Workshop* (IEEE Cat. No.01TH8584), 373-382.

Kabsch, W., and Sander, C., 'How good are predictions of protein secondary structure?', FEBS Letters, 155(2):179-82, 1983.

Kazemian HB, White K, Palmer-Brown D, Applications of evolutionary SVM to prediction of membrane alpha-helices, *Expert Systems with Applications* 40:3412–342, 2013.

Kazemian H, Yusuf SA, An ANFIS approach to transmembrane protein prediction. IEEE World Congress on Computational Intelligence (IEEE WCCI 2014), *IEEE International Conference on Fuzzy Systems,* Beijing China, pp.1360-1365, 2014.

Kazemian H.B, Yusuf S.A, White K (2014), Signal peptide discrimination and cleavage site identification using SVM and NN, Computers in Biology and Medicine 45:98–110

Kazemian H, Yusuf S.A, White K, and Grimaldi C.M (2016), NN approach and its comparison with NN-SVM to beta-barrel prediction, *Expert Systems with* Applications 61:203–214

Kazemian H and Grimaldi C.M (2018), Cascading classifier application for topology prediction of TMB proteins. *SSCI 2018*: 1049-1055

Kazemian H and Grimaldi C.M (2020), Cascading classifier application for topology prediction of transmembrane beta-barrel proteins, *Journal of Bioinformatics and Computational Biology*, Vol.18, No. 3

Kecman V., Huang, T. M., and Vogt. M. (2005). Iterative Single Data Algorithm for Training Kernel Machines from Huge Data Sets: Theory and Performance. *Support Vector Machines: Theory and Applications*. Edited by Lipo Wang, 255–274. Berlin: Springer-Verlag

Kedarisetti K.D, Kurgan L, Dick S. (2006). Classifier ensembles for protein structural class prediction with varying homology. *Biochemical and Biophysical Research Communications*. 348, 981–988.

Kégl, B. (2013). The Return of AdaBoost.MH: Multi-Class Hamming Trees. *arXiv:1312.6086 [cs]*, (preprint)

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C. and Murthy, K. R. K. (2001). Improvements to Platt's SMO Algorithm for SVM Classifier Design. *Neural Computing* 13, 637–49.

Khan, L., Awad, M., Thuraisingham, B. (2007) A new intrusion detection system using support vector machines and hierarchical clustering. VLDB J. 16(4), 507–521

Kim, H. and Park, H. (2003). Protein secondary structure prediction based on an improved support vector machines approach," *Protein Eng.*, vol. 16, no. 8, pp. 553–560.

Kim, S.E. and Seo, I.W. (2015). Artificial Neural Network ensemble modeling with conjunctive data clustering for water quality prediction in rivers. Journal of Hydro-environment Research, Volume 9(3), pp.325–339.

Koziol J.A., Feng A.C., Jia Z, Wang Y., Goodison S., McClelland M., et al. (2009). The wisdom of the commons: ensemble tree classifiers for prostate cancer prognosis. Bioinformatics.25, 54– 60.

Kong, E.B. and Dietterich, T.G. (1995). Error-Correcting Output Coding Corrects Bias and Variance. *Proceedings of the Twelfth International Conference on Machine Learning*, 313–321. Morgan Kaufmann.

Kraus, P. and Dzwinel, W. (2012). Nearest neighbor search by using Partial KD-tree method. Theoretical and Applied Informatics, Vol. 20, No. 3, pp.149–165

Kreßel, U. (1999). Advances in Kernel Methods. Edited by Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, 255–68. Cambridge, MA, USA: MIT Press, Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), pp.84–90

Krogh, A., Larsson, B., von Heijne, G. & Sonnhammer, E. L. (2001). Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. *Journal of Molecular Biol*ogy, 305, 567–80.

Kumar, S. and Apparao, M. (2017). A NOVEL APPROACH TO WAVELET-BASED FAULT CLASSIFICATION AND DETECTION FOR A TRANSMISSION LINE USING ARTIFICIAL NEURAL NETWORKS AND SUPPORT VECTOR MACHINE. *Original Article Journal of Electrical and Electronics Engineering (JEEE), Vol.*7, pp.1–14.

Kumar, V. P., and Poggio, T. (2000). Learning-Based Approach to Real Time Tracking and Analysis of Faces. *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, 96–101

Kuhn, H. W., and Tucker, A. W. (1951). *Nonlinear programming*. Proceedings of 2nd Berkeley Symposium. Berkeley: University of California Press. 481–492

Kuncheva, L.I. and Whitaker, C.J. (2003). Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine Learning* 51, 2

Kyte, J., and Doolittle, R. F. (1982). A Simple Method for Displaying the Hydropathic Character of a Protein. *Journal of Molecular Biology*,157,105–32.

Lang, K.J., Waibel, A.H., and Hinton, G.E. (1990). A Time-delay Neural Network Architecture for Isolated Word Recognition. *Neural Networks*, 3, 23–43.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep Learning. *Nature,* 521, 436–44.

Li, Wang, Zhang, Xin, and Liu (2019). Recurrent Neural Networks Based Photovoltaic Power Forecasting Approach. Energies, Vol. 12(13), p.2538.

Lin, C. F. and Wang, S. D. (2004). Training algorithms for fuzzy support vector machines with noisy data. Pattern Recognition Letters, 25, 1647-1656

Lin, C. F. and Wang, S. D. (2002) Fuzzy support vector machines. IEEE Transactions on Neural Networks 13, 464-471

Liu K.H, Xu C.G. (2009). A genetic programming-based approach to the classification of multiclass microarray datasets. *Bioinformatics.* 25,331–337.

Liu B., Cui Q., Jiang, T., Ma S. (2004). A combinational feature selection and ensemble neural network method for classification of gene expression data. *BMC Bioinformatics*.5, 136.

Lincoln, Y.S. & Guba, E.G. (2000), "Paradigmatic controversies, contradictions, and emerging influences". In N. Denzin and Y. Lincoln (eds.), Handbook of Qualitative Research (2nd ed., pp. 163-188). Thousand Oaks, CA: Sage

Lomize, M. A., Lomize, A. L., Pogozheva, I. D. & Mosberg, H. I. (2006). OPM: orientations of proteins in membranes database. Bioinformatics, 22, 623–5

Luo, Y. and Yang, F. (2014). Deep learning with noise.
https://pdfs.semanticscholar.org/d79b/a428e1cf1b8aa5d320a93166315bb30b4765.pdf

Magnan, C. N. and Baldi, P. (2014). SSpro/ACCpro 5: Almost Perfect Prediction of Protein Secondary Structure and Relative Solvent Accessibility Using Profiles, Machine Learning and Structural Similarity. *Bioinformatics*, 30, 2592–2597.

Martelli, P. L., Fariselli, P., Krogh, A. and Casadio, R. (2002). A sequence-profile-based HMM for predicting and discriminating beta barrel membrane proteins. *Bioinformatics*, 18, S46–53.

Mason, J. (2002) Qualitative Researching. 2nd Edition, Sage Publications, London

Mathworks (2016). Machine Learning with MATLAB. [Online] Available at: http://www.mathworks.com/solutions/machine-learning/ [Accessed 19 June. 2016].

Mathworks (2017). Classification ensembles. [Online] Available at: http://www.mathworks.com/ /help/stats/classification-ensembles.html [Accessed 20 Aug. 2017].

Melvin, I., Weston, J., Leslie, C.S and Noble, W.S. (2008). Combining Classifiers for Improved Classification of Proteins from Sequence or Structure. *BMC Bioinformatics,* 9, 389

Mitchell, M. (1997). Machine Learning, McGraw-Hill Computer science series.

Møller, M. F., "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning." Neural Networks 6, no. 4 (1993): 525–33

Mori, S., Suen, C., Yamamoto, K. (1995). Historical review of OCR research and development, pp.244–273. IEEE Computer Society Press, Los Alamitos

Mukherjee, S., Tamayo, P., Rogers, S., Rifkin, R., Engle, A., Campbell, C., Mesirov, J. P. 2003. Estimating dataset size requirements for classifying DNA microarray data. Journal of Computational Biology, 10(2), 119-142

Najafabadi, M.M, Villanustre, F., Khoshgoftaar, T.M., Seliya, N., Wald, R. and Muharemagic, E. (2015). Deep Learning Applications and Challenges in Big Data Analytics." *Journal of Big Data,* 2, 1

Nakajima, C., Pontil, M. and Poggio, T. (2000). People Recognition and Pose Estimation in Image Sequences. *Proceedings of the IEEE-INNS-ENNS International*

Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, 4, 189–194.

Natt, N. K., Kaur, H., Raghava, G. P. (2004). Prediction of transmembrane regions of beta-barrel proteins using ANN- and SVM-based methods. *Proteins*, 56, 11–18.

Neuman, W.L. (2003), "Social Research Methods: Qualitative and Quantitative Approaches" (5th ed.). Boston: Allyn and Bacon

Ng, A. (2015) Advice for applying machine learning. [Online]. Available at: https://see.stanford.edu/materials/aimlcs229/ML-advice.pdf
[Accessed 16 Jul. 2016]

Osuna, E., Freund, R. and Girosit, F. (1997). Training Support Vector Machines: An Application to Face Detection. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 130–36.

Ou, Y. Y., Chen, S. A and Gromiha, M. M. (2010). Prediction of Membrane Spanning Segments and Topology in Beta-Barrel Membrane Proteins at Better Accuracy. *Journal of Computational Chemistry* 31, 217–223.

Ouali, M., and King, R. D. (2000). Cascaded Multiple Classifiers for Secondary Structure Prediction." *Protein Science: A Publication of the Protein Society* 9, 6, 1162–76.

Paul, C., and Rosenbusch, J. P. (1985). Folding Patterns of Porin and Bacteriorhodopsin. *The EMBO Journal*, 4, 1593–97.

Pauling, L., and Corey, R. B. (1951). The Pleated Sheet, a New Layer Configuration of Polypeptide Chains." *Proceedings of the National Academy of Sciences of the United States of America*, 37, 251–56.

Pengyi, Y., Yee, H.Y., Zhou, B.B. and Zomaya, A.Y. (2010). A Review of Ensemble Methods in Bioinformatics. Current Bioinformatics, Vol. 5(4), pp.296–308

Pavlidis, P., Weston, J., Cai, J. and Noble, W.S. (2002). Learning Gene Functional Classifications from Multiple Data Types. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology* 9, 401–411.

Peng, Y. (2006) A Novel Ensemble Machine Learning for Robust Microarray Data Classification." *Computers in Biology and Medicine* 36, 553–573.

García-Pedrajas, N. and Ortiz-Boyer, D. (2011). An Empirical Study of Binary Classifier Fusion Methods for Multiclass Classification. *Information. Fusion* 12, 111–130.

Pittore, M., Basso, C. and Verri, A. (1999). Representing and Recognizing Visual Dynamic Events with Support Vector Machines. *Proceedings 10th International Conference on Image Analysis and Processing*, 18–23.

Platt, J. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. *Microsoft Research*, https://www.microsoft.com/enus/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-supportvector-machines/.

Pollastri, G., Przybylski, D., Rost, B. and Baldi, P. (2002). Improving the Prediction of Protein Secondary Structure in Three and Eight Classes Using Recurrent Neural Networks and Profiles. *Proteins: Structure, Function, and Bioinformatics*, 47, 228–235.

Platt, J.C., Cristianini, N. and Shawe-Taylor, J. (1999). Large Margin DAGs for Multiclass Classification. *Proceedings of the 12th International Conference on Neural Information Processing Systems*, 547–53. NIPS'99. Cambridge, MA, USA: MIT Press.

Pontil, M., and Verri, A. (1998). Support Vector Machines for 3D Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 637–46.

Poole, B., Sohl-Dickstein, J., and Ganguli, S. (2014). Analyzing noise in autoencoders and deep networks. *Cornell University Library*, arXiv:1406.1831

Powell, M.J.D. (1977). Restart procedures for the conjugate gradient method. *Mathematical Programming*, 12, 241–254.

Qian, N. and Sejnowski, T. J. "Predicting the secondary structure of globular proteins using neural network models," J. Mol. Biol., vol. 202, no. 4, pp. 865–884, Aug. 1988.

Rajendran, S.& Kalpana, B. (2011). Identifying Efficient Kernel Function in Multiclass Support Vector Machines. International Journal of Computer Applications, 28, 18-23

Raman, P., Cherezov, V. & Caffrey, M. (2006). The Membrane Protein Data Bank. Cellular and Molecular Life Sciences, 63, 36–51

Randall, A., Cheng, J., Sweredoski, M., and Baldi, P. (2008). TMBpro: Secondary Structure, BContact and Tertiary Structure Prediction of Transmembrane B-Barrel Proteins. *Bioinformatics*, 24, 513–20.

RCSB Protein Data Bank (2021). *RCSB PDB*. [online] Rcsb.org. Available at: https://www.rcsb.org/search/browse/cath [Accessed 12 Nov. 2021].

Riis, S. K. and Krogh, A. "Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments," *J. Comput. Biol.*, vol. 3, no. 1, pp. 163–183, 1996.

Robles V., Larranaga P., Pena J.M., Menasalvas E., Perez M.S., Herve, V., et al. (2004). Bayesian network multi-classifiers for protein secondary structure prediction. *Artificial Intelligence in Medicine.* 31, 117–136.

Romdhani, S., Schokopf, B. and Blake, A. (2001) Computationally efficient face detection. *Proceedings of International Conference Computer Vision*, 2, 695-700.

Russel, S. J. and Norvig, P. (2003). Artificial Intelligence: A Modern Approach. Prentice Hall, 3rd edition

Saier, M. H., Tran, C. V. and Barabote, R. D (2006). TCDB: The Transporter Classification Database for Membrane Transport Protein Analyses and Information. *Nucleic Acids Research* 34, D181–186.

Salamov, A. A. and Solovyev, V. V. "Protein secondary structure prediction using local alignments," *J. Mol. Biol.*, vol. 268, no. 1, pp. 31–36, Apr. 1997.

Schirmer, T. and Cowan, S. W. (1993). Prediction of membrane-spanning beta-strands and its application to maltoporin. *Protein Science*, 2, 1361–3.

Schölkopf, B., Burges, C.J.C., and Smola, A.J. (1999). *Advances in Kernel Methods: Support Vector Learning*. Cambridge, MA, USA: MIT Press.

Sharma, S., Sharma, V. (2016). Performance of Various Machine Learning Classifiers on Small Datasets with Varying Dimensionalities. Circulation in Computer Science, 1, 30-35.

Shen, H.B., and Chou, K.C. (2006). Ensemble Classifier for Protein Fold Pattern Recognition. *Bioinformatics* 22, 14, 1717–1722

Sietsma, J. and Dow, R. (1991). Creating artificial neural networks that generalize. *Neural Networks*, 4, 67–79

Singh, S., Singh, M. (2007). Progress in Pattern Recognition. Springer London

Singh, N., Goodman, A., Walter, P., Helms, V., and Hayat, S. (2011). TMB-HMM: A frequency profile based hmm for predicting the topology of transmembrane beta-barrel proteins and the exposure status of transmembrane residues. *Biochimica et Biophysica Acta (BBA)-Proteins & Proteomics*, 1814, 664–670.

Smola, A. J. and Schölkopf, B. (2004). A Tutorial on Support Vector Regression. *Statistics and Computing*, 3,199–222.

Smola, AJ., Schölkopf, B. and Müller, K.R. (1998). The Connection between Regularization Operators and Support Vector Kernels. *Neural Networks: The Official Journal of the International Neural Network Society* 11, 637–49.

Sollich, P. and Krogh, A. (1996). Learning with Ensembles: How Overfitting Can Be Useful. In *Advances in Neural Information Processing Systems 8*, edited by Touretzky, D.S, Mozer, M.C and Hasselmo, M.E, 190–96. MIT Press.

Sordo, M. and Zeng, Q. On Sample Size and Classification Accuracy: A Performance Comparison. *Biological and Medical Data Analysis*, 2005, pp. 193–201.

Spencer, M., Eickholt, J. and Cheng, J. (2015). A Deep Learning Network Approach to Ab Initio Protein Secondary Structure Prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics,* 12, 103–112.

Srihari, S. (n.d.). Deep Learning Srihari 1 Gradient-based Optimization. [online] Available at: https://cedar.buffalo.edu/~srihari/CSE676/4.2%20Gradient-based%20Optimization.pdf [Accessed 11 Nov. 2021].

Srinivasu, G., Rao, R.N., Nandy, T.K. and Bhattacharjee, A. (2012). Artificial Neural Network Approach for Prediction of Titanium Alloy Stress-Strain Curve. *Procedia Engineering*, Vol. 38, pp.3709–3714.

Suykens, J. A. K., De Brabanter, J., Lukas, L., and Vandewalle, J. (2002). Weighted least squares support vector machines: robustness and sparse approximation. Neurocomputing 48, 85105

Tama, A.B., Rhee, K.H., (2017). An extensive empirical evaluation of classifier ensembles for intrusion detection task. Computer Systems Science and Engineering. 32. 149-158.

Tanaka, S. and Scheraga, H. A. “Medium- and Long-Range Interaction Parameters between Amino Acids for Predicting Three-Dimensional Structures of Proteins,” *Macromolecules*, vol. 9, no. 6, pp. 945–950, Nov. 1976

Tang, Y. and Eliasmith, C. (2010). Deep networks for robust visual recognition. Proceedings of the 27th International Conference on Machine Learning, June 21-24, 2010, Haifa, Israel

Tavallaee, M., Bagheri, E., Lu, W. and Ghorbani, A.A. (2009). A detailed analysis of the KDD CUP 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications

Tay, F., and Cao, L.J. (2002). Modified support vector machines in financial time series forecasting, Neurocomputing, 48, 847-861

Teow, L.N., and Loe, K.F. (2002). Robust Vision-Based Features and Classification Schemes for off-Line Handwritten Digit Recognition. *Pattern Recognition*, 35, 2355–2364.

The NMITLI-BioSuite Team. (2007). BioSuite: A comprehensive bioinformatics software package (A unique industry–academia collaboration). *Current Science,* 92, 1.

The Stephen White Laboratory at UC Irvine, Department of Physiology and Biophysics, School of Medicine, University of California at Irvine [Online]. Available at: http://blanco.biomol.uci.edu/index.shtml [Accessed 03 May. 2016]

Topcons.net. (2021). *BOCTOPUS2: transmembrane beta-barrel topology prediction*. [online] Available at: https://b2.topcons.net/pred/download/ [Accessed 12 Nov. 2021].

Tsirigos, K. D, Bagos, P. G. and Hamodrakas, S. J. (2011). OMPdb: A database of ß-barrel outer membrane proteins from Gram negative bacteria. *Nucleic Acids Research*, 39, D324-D331

Tsujinishi, D., and Abe, S. (2003). Fuzzy least squares support vector machines for multiclass problems. Neural Networks, 16, 785-792. Advances in Neural Networks Research: IJCNN 2003.

Tusnady, G. E., Dosztanyi, Z. & Simon, I. (2005a). PDB TM: Selection and membrane localization of transmembrane proteins in the protein data bank. *Nucleic Acids Research*, 33, D275–D278.

Tusnady, G. E., Dosztanyi, Z. & Simon, I. (2005b). TMDET: web server for detecting transmembrane regions of proteins by using their 3D coordinates. *Bioinformatics*, 21, 1276–7.

Tusnady, G. E, Kalmár, L. & Simon, I. (2008). TOPDB: Topology Data Bank of Transmembrane Proteins. *Nucleic Acids Research*,36, D234-D239.

Vapnik, V. (1998) Statistical Learning Theory. Wiley, New York

Vapnik, V. (1999). *The Nature of Statistical Learning Theory*. 2nd edition, Springer.

Vilela, D.W.F.L., Ferreira, E.W.T., Shinoda, A.A., de Souza Araujo, N.V., de Oliveira, R. and Nascimento, V.E. (2014). A dataset for evaluating intrusion detection systems in IEEE 802.11 wireless networks. 2014 IEEE Colombian Conference on Communications and Computing (COLCOM).

Viola, P., and Jones, M. (2001) *Rapid Object Detection Using a Boosted Cascade of Simple Features*

Vogel, H. and Jahnig, F. (1986). Models for the structure of outer-membrane proteins of Escherichia coli derived from raman spectroscopy and prediction methods. *Journal of Molecular Biology*, 6, 191–199.

Von Heijne, G. (1992), Membrane protein structure prediction. Hydrophobicity analysis and the positive-inside rule. *J Mol Biol,*225.487–494.

Wan, V., and Campbell, W. M. (2000). Support Vector Machines for Speaker Verification and Identification. *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop*, 2, 775–784

Wang, G. and Dunbrack, R. L. (2003). PISCES: A Protein Sequence Culling Server. *Bioinformatics*, 19, 1589–1591.

Wang, S.Q., Yang, J., and Chou, K.C. (2006). Using Stacked Generalization to Predict Membrane Protein Types Based on Pseudo-Amino Acid Composition. *Journal of Theoretical Biology* 242, 4, 941–46.

Wang, T. Y., and Chiang, H. M. (2007). Fuzzy support vector machine for multi-class text categorization. Information Processing and Management, 43, 914-929.

Wang, Y., Chua, C.S. and Ho, Y.K. (2002). Facial Feature Detection and Face Recognition from 2D and 3D Images. *Pattern Recognition Letters* 23,10, 1191–1202.

Welte, W., Weiss, M. S., Nestel, U., Weckesser, J., Schiltz, E. and Schulz, G. E. (1991). Prediction of the General Structure of OmpF and PhoE from the Sequence and Structure of Porin from Rhodobacter Capsulatus. Orientation of Porin in the Membrane. *Biochimica Et Biophysica Acta* 1080, 271–74.

Wijaya E., Yiu S.M/, Son N.T., Kanagasabai R., Sung W.K. (2008). MotifVoter: a novel ensemble method for fine-grained integration of generic motif finders. *Bioinformatics*. 24, 2288– 2295.

Wikipedia Contributors (2019). Polynomial kernel. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Polynomial_kernel [Accessed 11 Nov. 2021].

William Stafford Noble (2017). 1 Support vector machine applications in computational biology. [online] Available at: https://www.semanticscholar.org/paper/1-Support-vector-machine-applications-in-biology-Noble/1b86be75e9c44351c8de387a3223e7930078c22c [Accessed 11 Nov. 2021].

Windeatt, T., and Ghaderi, R. (2003). *Coding and Decoding Strategies for Multi-Class Learning*, Information *Fusion,* 4, 1, 11–21

Wu, C.C, Asgharzadeh, S., Triche, T.J. and D'Argenio, D.Z. (2010). Prediction of Human Functional Genetic Networks from Heterogeneous Data Using RVM-Based Ensemble
Learning." *Bioinformatics*, 26, 807–813

Yang P., Zhou B., Zhang Z., Zomaya A. (2010a). A multi-filter enhanced genetic ensemble system for gene selection and sample classification of microarray data. *BMC Bioinformatics.* 11(Suppl 1), S5.

Yang P., Ho J.W.K., Zomaya A.Y., Zhou B.B. (2010b) A genetic ensemble approach for genegene interaction identification*. BMC Bioinformatics.* 11, 524.
Yang, Z.R., Chou, K.C.: Predicting the O-linkage sites in glycoproteins using bio-basis function neural networks. Bioinformatics 20, 903–908 (2004)

Yanover C., Weiss Y. (2004). Finding the AI Most Probable Configurations Using Loopy Belief Propagation. In: Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference. The MIT Press.289.

Yanover C., Singh M., Zaslavsky E. (2009). M are better than one: an ensemble-based motif finder and its application to regulatory element prediction. *Bioinformatics.*25, 868–874.

Zou, L., Wang, Z., Wang, Y. and Hu, F. (2010). Combined prediction of transmembrane topology and signal peptide of β-barrel proteins: Using a hidden Markov model and genetic algorithms. *Computers in Biology and Medicine*,40,621-628

Appendix A

Accuracy (%)	Window Size	Bits encoding	Transfer functon	Hidden layer size	Training function	Performance Fcn	Data division	Dataset used
64.5	53	20	logsig	70	trainscg	sum squared error	Dividerand	TopBPdataset
63.1	54	20	logsig	70	trainscg	sum squared error	Dividerand	TopBPdataset
64.3	52	20	logsig	70	trainscg	sum squared error	Dividerand	TopBPdataset
63.8	52	20	logsig	68	trainscg	sum squared error	Dividerand	Boctopus2
62.3	54	20	logsig	70	trainscg	crossentropy	Dividerand	Boctopus2
63.7	54	20	logsig	68	trainscg	sum squared error	Dividerand	Boctopus2
60.9	51	20	logsig	72	trainscg	sum squared error	Dividerand	Boctopus2
64.5	53	20	logsig	70	trainscg	sum squared error	Dividerand	Boctopus2
64.5	53	20	tansig	70	trainscg	sum squared error	Dividerand	Boctopus2
53.2	53	20	logsig	70	trainscg	Sum absolute error performance function	Dividerand	TopBPdataset
64	53	20	logsig	70	trainscg	Mean squared normalized error performance function	Dividerand	TopBPdataset
53.2	53	20	logsig	70	trainscg	Mean absolute error performance function	Dividerand	TopBPdataset
63.8	53	20	logsig	70	traincgp	sum squared error	Dividerand	TopBPdataset
58.6	53	20	logsig	70	traincgb	sum squared error	Dividerand	TopBPdataset
53.4	50	20	logsig	70	trainrp	sum squared error	Dividerand	TopBPdataset
62.9	79	20	logsig	70	trainscg	sum squared error	Dividerand	TopBPdataset
54.6	80	20	logsig	70	trainscg	sum squared error	Dividerand	TopBPdataset
60.2	20	20	logsig	70	trainscg	sum squared error	Dividerand	TopBPdataset
33.1	70	20	logsig	70	traingdx	sum squared error	Dividerand	TopBPdataset
45.9	20	20	logsig	70	traingdx	sum squared error	Dividerand	TopBPdataset
53.2	53	20	logsig	70	traindgx	sum squared error	Dividerand	TopBPdataset
48.5	53	20	logsig	70	traincgf	sum squared error	Dividerand	TopBPdataset
63	53	20	logsig	73	trainscg	sum squared error	Dividerand	TopBPdataset

Appendix B

Accuracy (%)	K Value (NumNeighbors)	Nearest Neighbor search method (Nsmethod)	Tie-breaking algorithm (BreakTies)	Maximum data points in node (Bucketsize)	Tie inclusion flag (IncludeTies)	Distance (Distance)	Exponent (Exponent)	Data division	Dataset used	Comments
46.2	2	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	77.4-I; 32.0-O; 15.1-M
48.6	4	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	71.9-I; 33.0-O; 29.7-M
51.8	8	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	76.3-I; 33.3-O; 32.0-M
51.9	16	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	79.1-I; 29.7-O; 32.5-M
53.3	32	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	85.6-I; 26.7-O; 30.4-M
52.2	64	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	89.4-I; 20.7-O; 26.7-M
50.5	128	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	94.2-I; 13.0-O; 20.9-M
48.5	256	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	96.5-I; 7.6-O; 15.6-M
45.8	512	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	98.8-I; 2.6-O; 8.2-M
44.3	1024	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	99.6-I; 0.4-O; 4.0-M
53.1	42	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	87.4-I; 23.3-O; 30.1-M
53.3	29	exhaustive	random	n/a	n/a	n/a	n/a	79% for training	Boctopus2	84.6-I; 27.6-O; 31.4-M
52.8	29	exhaustive	random	n/a	n/a	n/a	n/a	70% for training	Boctopus2	83.9-I; 26.6-O; 30.8-M
51.7	29	exhaustive	random	n/a	n/a	n/a	n/a	60% for training	Boctopus2	83.2-I; 25.7-O; 29.5-M
53.6	29	exhaustive	random	n/a	n/a	n/a	n/a	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
53.1	29	exhaustive	random	n/a	n/a	n/a	n/a	90% for training	Boctopus2	85.7-I; 25.9-O; 30.6-M
52.9	29	exhaustive	random	n/a	n/a	n/a	n/a	87% for training	Boctopus2	85.2-I; 27.1-O; 30.2-M
53.6	29	kdtree	random	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
52.9	64	kdtree	random	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	89.9-I; 21.3-O; 27.1-M
51.4	128	kdtree	random	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	94.2-I; 14.5-O; 21.8-M
49.1	256	kdtree	random	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	96.7-I; 7.8-O; 16.4-M
53.1	29	kdtree	random	n/a	n/a	euclidean	n/a	90% for training	Boctopus2	85.7-I; 25.9-O; 30.6-M
52.8	29	kdtree	random	n/a	n/a	euclidean	n/a	70% for training	Boctopus2	83.9-I; 26.6-O; 30.8-M
53.6	29	kdtree	random	n/a	n/a	cityblock	n/a	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
52.9	64	kdtree	random	n/a	n/a	cityblock	n/a	85% for training	Boctopus2	89.9-I; 21.3-O; 27.1-M
51.4	128	kdtree	random	n/a	n/a	cityblock	n/a	85% for training	Boctopus2	94.2-I; 14.5-O; 21.8-M
53.6	29	kdtree	random	n/a	n/a	minkoswki	2	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
52.9	64	kdtree	random	n/a	n/a	minkoswki	2	85% for training	Boctopus2	89.9-I; 21.3-O; 27.1-M
51.4	128	kdtree	random	n/a	n/a	minkoswki	2	85% for training	Boctopus2	94.2-I; 14.5-O; 21.8-M
43.3	29	kdtree	random	n/a	n/a	chebychev	n/a	85% for training	Boctopus2	100-I; 0-O; 0-M
43.3	64	kdtree	random	n/a	n/a	chebychev	n/a	85% for training	Boctopus2	100-I; 0-O; 0-M
43.3	128	kdtree	random	n/a	n/a	chebychev	n/a	85% for training	Boctopus2	100-I; 0-O; 0-M
53.6	29	kdtree	smallest	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
52.9	64	kdtree	smallest	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	89.9-I; 21.3-O; 27.1-M
51.4	128	kdtree	smallest	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	94.2-I; 14.5-O; 21.8-M
53.6	29	kdtree	nearest	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
52.9	64	kdtree	nearest	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	89.9-I; 21.3-O; 27.1-M
51.4	128	kdtree	nearest	n/a	n/a	euclidean	n/a	85% for training	Boctopus2	94.2-I; 14.5-O; 21.8-M
52.7	29	exhaustive	n/a	n/a	true.	n/a	n/a	85% for training	Boctopus2	91.2-I; 20.9-O; 25.0-M
51.8	64	exhaustive	n/a	n/a	true.	n/a	n/a	85% for training	Boctopus2	93.6-I; 16.5-O; 22.5-M
48.3	128	exhaustive	n/a	n/a	true.	n/a	n/a	85% for training	Boctopus2	97.0-I; 8.3-O; 13.3-M
52.7	29	kdtree	n/a	n/a	true.	euclidean	n/a	85% for training	Boctopus2	91.2-I; 20.9-O; 25.0-M
52.7	29	kdtree	n/a	40	true.	euclidean	n/a	85% for training	Boctopus2	91.2-I; 20.9-O; 25.0-M
52.7	29	kdtree	n/a	80	true.	euclidean	n/a	85% for training	Boctopus2	91.2-I; 20.9-O; 25.0-M
52.7	29	kdtree	n/a	580	true.	euclidean	n/a	85% for training	Boctopus2	91.2-I; 20.9-O; 25.0-M
52.7	29	kdtree	n/a	580	true.	minkoswki	2	85% for training	Boctopus2	91.2-I; 20.9-O; 25.0-M
52.7	29	kdtree	n/a	10	true.	minkoswki	2	85% for training	Boctopus2	91.2-I; 20.9-O; 25.0-M
53.6	29	kdtree	random	10	n/a	minkoswki	2	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
53.6	29	kdtree	random	100	n/a	minkoswki	2	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
53.6	29	kdtree	random	100	n/a	minkoswki	3	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
53.6	29	kdtree	random	100	n/a	minkoswki	30	85% for training	Boctopus2	85.1-I; 28.4-O; 30.5-M
63.4	2	exhaustive	random	n/a	n/a	n/a	n/a	85% for training	TopBPdataset	76.8-I; 65.5-O; 56.4-M
66.9	4	exhaustive	random	n/a	n/a	n/a	n/a	85% for training	TopBPdataset	59.7-I; 60.8-O; 72.7-M
67.4	8	exhaustive	random	n/a	n/a	n/a	n/a	85% for training	TopBPdataset	49.7-I; 54.2-O; 80.8-M
66.7	16	exhaustive	random	n/a	n/a	n/a	n/a	85% for training	TopBPdataset	37.2-I; 43.5-O; 89.5-M
64.1	32	exhaustive	random	n/a	n/a	n/a	n/a	85% for training	TopBPdataset	25.8-I; 32.5-O; 94.4-M
60.3	64	exhaustive	random	n/a	n/a	n/a	n/a	85% for training	TopBPdataset	14.4-I; 18.9-O; 98.1-M
67.6	8	exhaustive	random	n/a	n/a	n/a	n/a	90% for training	TopBPdataset	49.7-I; 54.1-O; 81.2-M
68.6	8	exhaustive	random	n/a	n/a	n/a	n/a	95% for training	TopBPdataset	52.9-I; 54.8-O; 81.5-M
69.7	8	exhaustive	random	n/a	n/a	n/a	n/a	97% for training	TopBPdataset	52.5-I; 47.8-O; 87.0-M
71.8	8	exhaustive	random	n/a	n/a	n/a	n/a	99% for training	TopBPdataset	57.7-I; 56.5-O; 83.8-M
66.6	8	exhaustive	random	n/a	n/a	n/a	n/a	80% for training	TopBPdataset	47.9-I; 53.2-O; 80.6-M
67.6	8	kdtree	random	n/a	n/a	euclidean	n/a	90% for training	TopBPdataset	49.9-I; 54.1-O; 81.2-M
66.7	16	kdtree	random	n/a	n/a	euclidean	n/a	90% for training	TopBPdataset	38.5-I; 43.7-O; 89.0-M
67.6	8	kdtree	random	n/a	n/a	cityblock	n/a	90% for training	TopBPdataset	49.9-I; 54.1-O; 81.2-M
67.6	8	kdtree	random	n/a	n/a	minkoswki	2	90% for training	TopBPdataset	49.9-I; 54.1-O; 81.2-M
54.7	8	kdtree	random	n/a	n/a	chebychev	n/a	90% for training	TopBPdataset	2.3-I; 2.7-O; 100-M
67.6	8	exhaustive	smallest	n/a	n/a	n/a	n/a	90% for training	TopBPdataset	49.7-I; 54.1-O; 81.2-M
67.6	8	exhaustive	nearest	n/a	n/a	n/a	n/a	90% for training	TopBPdataset	49.7-I; 54.1-O; 81.2-M
66.8	8	exhaustive	n/a	n/a	true.	n/a	n/a	90% for training	TopBPdataset	36.9-I; 43.7-O; 89.9-M
63.1	16	exhaustive	n/a	n/a	true.	n/a	n/a	90% for training	TopBPdataset	24.0-I; 30.4-O; 94.4-M
60.8	32	exhaustive	n/a	n/a	true.	n/a	n/a	90% for training	TopBPdataset	15.5-I; 21.8-O; 97.6-M
66.8	8	kdtree	n/a	40	true.	euclidean	n/a	90% for training	TopBPdataset	36.9-I; 43.7-O; 89.9-M
66.8	8	kdtree	n/a	80	true.	euclidean	n/a	90% for training	TopBPdataset	36.9-I; 43.7-O; 89.9-M
66.8	8	kdtree	n/a	10	true.	minkoswki	2	90% for training	TopBPdataset	36.9-I; 43.7-O; 89.9-M
66.8	8	kdtree	n/a	20	true.	minkoswki	8	90% for training	TopBPdataset	36.9-I; 43.7-O; 89.9-M

Appendix C

Accuracy (%)	Kernel function(KernelFunction)	Store support vectors, their labels and α coefficients (SaveSupportVectors)	Box constraint (BoxConstraint)	Cache size (CacheSize)	Flag to clip alpha coefficients (ClipAlphas)	Optimization routing (Solver)	Tolerance for gradient difference (DeltaGradientTolerance)	Feasibility gap tolerance (GapTolerance)	Maximal number of numerical optimization iterations (IterationLimit)	Kernel offset parameter(KernelOffset)	Kernel scale parameter (KernelScale)	Karush-Kuhn-Tucker complementarity conditions violation tolerance (KKTTolerance)	ν parameter for one-class learning (Nu)	Number of iterations between optimization diagnostic message output (NumPrint)	Expected proportion of outliers in training data (OutlierFraction)	Polynomial kernel function order (PolynomialOrder)	Number of iterations between movement of observations from active to inactive set (ShrinkagePer	Flag to standardize predictor data(Standardize)	Verbosity level (Verbose)	Dataset
60.1	linear	false(default)	1(default)	1000(default)	true(default)	SMO(default)	0.001	0(default)	1e6(default)	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
60.1	linear	true	1(default)	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	1e6(default)	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
43.3	rbf	true	1(default)	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	1e6(default)	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
39	linear	true	1(default)	10000	true(default)	SMO(default)	0.001(SMO default)	0(default)	100	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
39.8	linear	true	1(default)	10000	true(default)	SMO(default)	0.001(SMO default)	0(default)	1000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
41.2	linear	true	1(default)	10000	true(default)	SMO(default)	0.001(SMO default)	0(default)	10000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
39	linear	true	1(default)	500	true(default)	SMO(default)	0.001(SMO default)	0(default)	100	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
38.5	linear	true	1(default)	500	true(default)	SMO(default)	0.001(SMO default)	0(default)	500	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
60.2	linear	true	1(default)	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
40.7	linear	true	100	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
60.2	linear	true	0.5	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
60.5	linear	true	0.1	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
58.4	linear	true	0.001	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
60.5	linear	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
60.4	linear	true	0.1	1000(default)	false	ISDA	0(ISDA default)	0(default)	100000	0 .1(default for ISDA)	1(default)	1e-3 (default for ISDA)	0.5(default)	1000(default)	0.05	n/a	0(default)	false(default)	0(default)	Boctopus2
59.8	linear	true	0.1	1000(default)	false	ISDA	0(ISDA default)	0(default)	100000	0 .1(default for ISDA)	1(default)	1e-3 (default for ISDA)	0.5(default)	1000(default)	0.95	n/a	0(default)	false(default)	0(default)	Boctopus2
25	linear	true	0.1	1000(default)	false	SMO(default)	0.01	0(default)	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
43.2	linear	true	0.1	1000(default)	false	SMO(default)	0.0001	0(default)	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
54	linear	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0.5	100000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
43.2	linear	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0.5	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
60.3	linear	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	0.5	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	Boctopus2
62.1	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	0.5	0 (default for SMO)	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
62.5	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	0.95	0 (default for SMO)	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
62.6	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0 (default for SMO)	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
59.9	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	2.5	10	0 (default for SMO)	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
59.9	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	10	0 (default for SMO)	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
62.1	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	auto	0 (default for SMO)	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
62.5	polynomial	true	0.1	1000(default)	false	ISDA	0(ISDA default)	0(default)	100000	0 .1(default for ISDA)	0.95	1e-3 (default for ISDA)	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
44.1	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	10	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
62.6	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0.00001	0.5(default)	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
62.6	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0.00001	0.25	1000(default)	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
62.6	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0.00001	0.5(default)	500	0(default)	3(default)	0(default)	false(default)	0(default)	Boctopus2
62.6	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0.00001	0.5(default)	500	0(default)	3(default)	0(default)	false(default)	1	Boctopus2
64.8	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0.00001	0.5(default)	500	0(default)	5	0(default)	false(default)	0(default)	Boctopus2
58.1	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0.00001	0.5(default)	500	0(default)	15	0(default)	false(default)	0(default)	Boctopus2
63.8	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0.00001	0.5(default)	500	0(default)	5	1000	false(default)	0(default)	Boctopus2
52	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	100000	0 (default for SMO)	5	0.00001	0.5(default)	500	0(default)	5	1000	true	0(default)	Boctopus2
63.8	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	1500000	0 (default for SMO)	5	0.00001	0.5(default)	500	0(default)	5	0(default)	false(default)	0(default)	Boctopus2
63.8	polynomial	true	0.1	1000(default)	false	SMO(default)	0.001(SMO default)	0(default)	2000000	0 (default for SMO)	5	0.00001	0.5(default)	500	0(default)	5	0(default)	false(default)	0(default)	Boctopus2
48	linear	false(default)	1(default)	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	100	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	TopBPdataset
47.7	linear	false(default)	1(default)	1000(default)	true(default)	SMO(default)	0.001(SMO default)	0(default)	1000	0 (default for SMO)	1(default)	0 (default for SMO)	0.5(default)	1000(default)	0(default)	n/a	0(default)	false(default)	0(default)	TopBPdataset

Appendix D

Configurations: **1**:Layer 1(KNN=1,SVM=1,DNN=1)/Layer 2 (SVM) **2**:Layer 1(KNN=0,SVM=1,DNN=1)/Layer 2 (SVM) **3**:Layer 1(KNN=1,SVM=0,DNN=1)/Layer 2 (SVM) **4**:Layer 1(KNN=1,SVM=1,DNN=0)/Layer 2 (SVM)	Configuration 3	Configuration 3	Configuration 3	Configuration 3	Configuration 3	Configuration 3	Configuration 3	Configuration 3	Configuration 4	Configuration 4	Configuration 1	Configuration 1	Configuration 2	Configuration 1	Configuration 1
Dataset	TopBPDataset	TopBPDataset	TopBPDataset	TopBPDataset	TopBPDataset	TopBPDataset	TopBPDataset	TopBPDataset	TopBPDataset	TopBPDataset	TopBPDataset	Boctopus2dataset	TopBPDataset	TopBPDataset	TopBPDataset
tspart1	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.75	0.8
tspart2	0.6	0.9	0.75	0.35	0.25	0.4	0.45	0.42	0.42	0.42	0.42	0.42	0.42	0.42	0.42
Overall accuracy	61.2	49.4	53.1	65.2	64.5	66.1	65.4	66.2	72.8	71.9	73.1	63.6	72.8	74.6	75.2
KernelFunction									polynomial	polynomial	polynomial	polynomial	polynomial	polynomial	polynomial
SaveSupportVectors									TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
BoxConstraint									1(default)	0.1	1(default)	1(default)	1(default)	1(default)	1(default)
CacheSize									1000(default)	1000(default)	1000(default)	1000(default)	1000(default)	1000(default)	1000(default)
ClipAlphas									true(default)	FALSE	true(default)	true(default)	true(default)	true(default)	true(default)
Solver									SMO(default)	SMO(default)	SMO(default)	SMO(default)	SMO(default)	SMO(default)	SMO(default)
DeltaGradientTolerance									1e-3(default)	1e-3(default)	1e-3(default)	1e-3(default)	1e-3(default)	1e-3(default)	1e-3(default)
GapTolerance									0(default)	0(default)	0(default)	0(default)	0(default)	0(default)	0(default)
IterationLimit									1e6(default)	100000	1e6(default)	1e6(default)	1e6(default)	1e6(default)	1e6(default)
KernelOffset									0	0	0	0	0	0	0
KernelScale									1(default)	5	1(default)	1(default)	1(default)	1(default)	1(default)
KKTTolerance									0	0.00001	0	0	0	0	0
Nu									0.5(default)	0.5(default)	0.5(default)	0.5(default)	0.5(default)	0.5(default)	0.5(default)
NumPrint									1000(default)	500	1000(default)	1000(default)	1000(default)	1000(default)	1000(default)
OutlierFraction									0(default)	0(default)	0(default)	0(default)	0(default)	0(default)	0(default)
PolynomialOrder									3 (default)	5	3 (default)	3 (default)	3 (default)	3 (default)	3 (default)
ShrinkagePeriod									0(default)	0(default)	0(default)	0(default)	0(default)	0(default)	0(default)
Standardize									false(default)	false(default)	false(default)	false(default)	false(default)	false(default)	false(default)
Verbose									0(default)	0(default)	0(default)	0(default)	0(default)	0(default)	0(default)
NumNeighbors	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
Nsmethod	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive	exhaustive
BreakTies	random	random	random	random	random	random	random	random	random	random	random	random	random	random	random
Bucketsize	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
IncludeTies	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Distance	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Exponent	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Window Size	65	65	65	65	65	65	65	65			65	65	65	65	65
Bits encoding	50	50	50	50	50	50	50	50			50	50	50	50	50
Transfer functon	logsig	logsig	logsig	logsig	logsig	logsig	logsig	logsig			logsig	logsig	logsig	logsig	logsig
Hidden layer size	50	50	50	50	50	50	50	50			50	50	50	50	50
Training function	trainscg	trainscg	trainscg	trainscg	trainscg	trainscg	trainscg	trainscg			trainscg	trainscg	trainscg	trainscg	trainscg
Performance Function	sum squarred error	sum squarred error	sum squarred error	sum squarred error	sum squarred error	sum squarred error	sum squarred error	sum squarred error			sum squarred error	sum squarred error	sum squarred error	sum squarred error	sum squarred error
Data division	Dividerand	Dividerand	Dividerand	Dividerand	Dividerand	Dividerand	Dividerand	Dividerand			Dividerand	Dividerand	Dividerand	Dividerand	Dividerand